NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

CODE OPTIMIZATION AND HIERARCHICAL SCHEDULING TECHNIQUES FOR IMPLEMENTING THE COMMANDER PATROL WING TEN, MOFFETT FIELD, CALIFORNIA (COMPATWING TEN) TRAINING SCHEDULE ON MICROCOMPUTERS

by

Robert D. Powell

June 1989

Thesis Advisor: Neil C. Rowe

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: UNCLASSIFIED	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
2b Declassification/Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (*If Applicable*): 52	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization; 8b Office Symbol (*If Applicable*)	9 Procurement Instrument Identification Number
8c Address (*city, state, and ZIP code*)	10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No

11 Title (*Include Security Classification*)
CODE OPTIMIZATION AND HIERARCHICAL SCHEDULING TECHNIQUES FOR IMPLEMENTING THE COMMANDER PATROL WING TEN, MOFFETT FIELD, CALIFORNIA (COMPATWING TEN) TRAINING SCHEDULE ON MICROCOMPUTERS

12 Personal Author(s)
Powell, Robert D.

13a Type of Report	13b Time Covered	14 Date of Report (*year, month,day*)	15 Page Count
Master's Thesis	From To	June 1989	128

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	Schedule, Artificial Intelligence, Prolog, Training, Air Wing, Decision Support, Knowledge Base, Search.

19 Abstract (*continue on reverse if necessary and identify by block number*)

This research examines the practicality of using microcomputers and heuristic search techniques to handle scheduling problems. A program was developed using a hierarchical approach to produce an Annual Training Schedule for Commander Patrol Wing Ten, which includes the ready-alerts and six major inspections completed by each of the seven squadrons prior to deployments. The scheduling process is broken into three major phases: (a) Initialize the program database, (b) Determine the optimal month to schedule each inspection, and (c) Determine the optimal sequence of days within the month to schedule each inspection. The program is written in the language M-Prolog and runs on a Motorola 68020-based workstation. Comparisons made between the manually produced and the computer-generated schedules using data for two different years show that a microcomputer is capable of producing a more optimal schedule in much less time.

20 Distribution/Availability of Abstract: [X] unclassified/unlimited [ ] same as report [ ] DTIC users	21 Abstract Security Classification: UNCLASSIFIED
22a Name of Responsible Individual: Prof. Neil C. Rowe	22b Telephone (*Include Area code*): (408) 646-2462; 22c Office Symbol: Code 52Rp

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted

All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

Code Optimization and Hierarchical Scheduling Techniques for Implementing the Commander Patrol Wing Ten, Moffett Field, California (COMPATWING TEN) Training Schedule on Micrcomputers

by

Robert David Powell
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June1989

Author: Robert Powell
Robert David Powell

Approved by: Neil C. Rowe
Neil C. Rowe, Thesis Advisor

Yuh-jeng Lee, Second Reader

Robert B. McGhee, Chairman, Department of Computer Science

K. T. Marshall
Kneale T. Marshall, Dean of Information and Policy Sciences

ABSTRACT

This research examines the practicality of using microcomputers and heuristic search techniques to handle scheduling problems. A program was developed using a hierarchical approach to produce an Annual Training Schedule for Commander Patrol Wing Ten, which includes the ready-alerts and six major inspections completed by each of the seven squadrons prior to deployments. The scheduling process is broken into three major phases: (a) Initialize the program database, (b) Determine the optimal month to schedule each inspection, and (c) Determine the optimal sequence of days within the month to schedule each inspection. The program is written in the language M-Prolog and runs on a Motorola 68020-based workstation. Comparisons made between the manually produced and the computer-generated schedules using data for two different years show that a microcomputer is capable of producing a more optimal schedule in much less time.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

Most assignments and operational commitments during a professional Naval Officer's typical career require inordinate amounts of time separated from family and loved ones. The preparation of this thesis ensured that the assignment to the Naval Postgraduate School was not an exception for me. I wish to express my deepest and most sincere appreciation to my wife, Sherry, and our three daughters for their special understanding and loving support during the past nine months.

I. INTRODUCTION

A. MILITARY PERSONNEL TURNOVER

Military organizations have a continuous flow of personnel coming in and going out. Regardless of the fluctuating manning level, responsibilities must be kept and commitments must be fulfilled by these organizations. Before leaving, the outgoing personnel do what they can to pass along to those they leave behind the knowledge and techniques they have discovered along the way . Those that are filling the shoes of the experienced must often learn quickly and frequently while enduring the pressures from superiors to produce immediate results.

Turnover files, job description notebooks, and even one-on-one pass-downs cannot always prepare the newly assigned personnel for the decisions they must make. The more complex a job or task becomes, the harder it is to learn competently in a short period of time. Without the benefit of experience or time on the job the new personnel must rely heavily on their past experiences and intelligence to solve the new problems. Many decisions involve very complex situations with multitudes of details that are easily overlooked or forgotten even by the experienced personnel. With the advent of the micro-computer, assistance need not go with the transferred personnel. Programs that assist in solving problems, that do not need to be retaught the complexities involved, and that will not forget the miniscule details that are so easily overlooked during the rush to complete a task must be made available.

Developing and maintaining the schedule of activities for assets is one example area in which programs need to be developed. The emphasis of this thesis has been to design a system to be used on a microcomputer by the training officer at Patrol Wing Ten, Moffett Field, California (CPW-10) to produce a yearly schedule of the at-home inspections required of the seven squadrons stationed there.

B. SELECTED SCHEDULING PROBLEM

Scheduling problems have qualities that are generic and yet there invariably are variables and exceptions that make each a unique problem. This prevents a generic solution to the problems of scheduling. Scheduling the training of the seven Moffett squadrons involves managing not only the squadrons, but also the inspection teams. Time constraints are added externally and internally. Operational commitments must be allowed for. Inspection team availability must be considered. Qualification time limits cannot be exceeded.

The training officer is computationally overwhelmed when attempting to optimize such a schedule. There are simply too many possibilities for any one person to consider them all. Even microcomputers become over-taxed with the combinatorial problem that arises when attempting to find the best way to schedule six different activities for seven different squadrons over a twelve-month period.

In order to solve the combinatorial problems, this thesis has approached scheduling in a hierarchical manner. Heuristics are used to select the month that each inspection should ideally take place. Conflicts that occur between inspections with the same ideal month are resolved by finding a new month to hold one of the inspections within a three month window either side of its ideal month. Once a month is selected for all of the inspections that are to be scheduled, the optimum trialperiod for each event within its particular month can be found. The original combinatorial problem is thus greatly reduced, allowing the overall scheduling process to be done quicker and using less space.

C. DESCRIPTION OF REMAINING CHAPTERS

Background information on general scheduling algorithms is discussed in chapter II. In chapter III the details of the scheduling problem selected for this thesis are amplified. The rules used by the Training Officer to manually compute the yearly schedule will be explained. The algorithms used in the past to solve the CPW-10 training schedule problem are also outlined in chapter III along with the approach taken by LCDR David Hutson to computerize this scheduling.

The approach this thesis undertook to solve the CPW-10 training schedule problem is explained in chapter IV. A comparison of the improvements it made over previous techniques and the details of the M-Prolog code that was developed in this thesis are examined in chapter IV. The results achieved during this thesis are covered in chapter V. Recommendations and a summary of this thesis can be found in chapter VI.

Appendix A holds the actual source code for this program. Appendix B contains a sample data file. An example of the file containing a final schedule produced is found in Appendix C.

II. BACKGROUND

A. SCHEDULING

Scheduling is something people do to organize the way they intend to use the resources at their disposal to achieve their goals. More specifically, it is the ordering of the steps necessary to achieve goals, considering the start and/or finish times of the steps. The similarities of these two terms causes them to commonly be used synonymously. In many situations if a person determines the sequence of events the schedule will automatically fall into place. This occurs in situations where the events begin at a particular spot during the period selected for it, such as the earliest time or the latest time possible. However, sequencing does not take into account the idle time between events and scheduling does.[Ref. 1, p.9]

Though scheduling has been practiced for centuries, it was not formalized until the early 1900's. The Gnatt chart, developed by Henry L. Gnatt, used in production scheduling during World War I became a standard tool for schedulers.[Ref. 2] This technique's simplicity and graphical appeal are the reasons it is still commonly used today.

During the late 1950's with the advent of the electronic computer, new scheduling techniques emerged. Two significant network-based models that were developed during that period were the Critical Path Method (CPM) and the Performance Evaluation and Review Technique (PERT) [Ref. 2]. These two methods fall in the category of project scheduling, planning of activities that must be given a precedence with or without resource constraints. PERT in general finds the schedule that sets the start and finish times resulting in the minimum project time. CPM finds a schedule having the lowest cost during a specified time. Both of these methods used techniques based on the solutions of theoretical problems such as the shortest-route problem, the critical path problem, or the flow problem on a network. [Ref. 3, pp. 48-50]

B. AI TECHNIQUES

A scheduling problem that is solved using CPM can be translated into a heuristic search. Accomplishing a task or a machine completing its work becomes a transition from one state to another. A state is a group of events that have been scheduled. Each state will have costs associated with the arrangement of the events within it. For very simple problems every possible combination of transitions can be determined, to yield all the schedules possible, and to then select the one with the least cost as the solution.

But in most "real-world" scheduling, the size of the combinatorial problem is too large to be accomplished in a reasonable amount of time. Artificial-intelligence search techniques such as A*, hill-climbing, and simulated annealing have been used to solve scheduling problems. These methods use heuristics to guide the search to an optimum solution and avoid the combinatorial problem.

The A* search technique uses an agenda of states from which it repeatedly chooses the one with the best evaluation and total cost sum to work with. It then finds all the possible new states, or successors, which can be reached from the selected best-state using a single transition. Each of these successors, together with their evaluation and total cost, is added to the agenda. This process repeats until a state selected from the agenda is the goal.

The agenda maintained in the A* search can grow very large, consuming considerable amounts of computer memory. For this reason the no-path search is sometimes a preferred search algorithm. This method is a modification to the A* search, developed by Prof. Rowe [Ref. 4]. This algorithm varies from the A* by not storing state sequences used to reach agenda states, and pruning the agenda after each cycle. A "k-factor" can be set to determine how much of the agenda will be pruned. Only items whose evaluations are a k less than the best-state's evaluation are maintained on the agenda. This can sometimes prevent finding the absolute optimum path, but a large agenda can be prevented.

The hill-climbing search is a depth-first search that uses an evaluation function. With this search no agenda at all is maintained. This method does not allow backtracking to try alternate routes and will only work in situations for which the goal is guaranteed reachable from any of the possible routes selected.

Simulated annealing is like hill-climbing but uses stochastic processes to help choose from the agenda. In a scheduling situation the process starts with an initial schedule. Events are then randomly repositioned. The result is evaluated after each repositioning. The process continues until the resulting schedule cost reaches an asymptote. This method does not guarantee the absolute optimized schedule but has been found effective finding improved schedules and can minimize the combinatorial problem. [Ref. 5,6].

III. APPLICATION DESCRIPTION

Of the information used in developing this program not taken from [Ref. 7], most was gathered during interviews with the officers in the CPW-10 Training Department [Ref. 8,9,10]. The author was also able to draw on his own experience gained during prior operational tours at both the squadron and wing levels.

A. SQUADRON CYCLE

The CPW-10 Training Officer is responsible for the training of the seven Patrol Squadrons permanently stationed at Naval Air Station Moffett Field, California. Of the seven squadrons, two are on operational deployments any given time. The emphasis of the training schedule is to prepare the at-home squadrons to be ready for deployment. The normal cycle for the squadrons is twelve months at-home and six months on deployments. This can be changed at any time but is a general rule.

During the at-home period the squadrons are given a thirty day period immediately following the return from deployment called the Post-deployment Safety Stand-down period. This period is given to the squadron to basically regroup after the stresses brought upon by operational deployments. The training officer avoids scheduling any events for a squadron during its Safety Stand-down period.

Ready-alert periods are another time interval in which the training officer avoids scheduling any events for the squadrons. The ready-alert periods are very similar to a deployment period. The major difference is that the squadron remains at-home. The ready-alert periods are also normally only a month in duration. On some occasions it is necessary to extend a ready-alert period to one and a half months. During a ready-alert period a squadron must be prepared to assume any operational tasking that might arise. Operational tasking would take priority over and would prevent the completion of any inspection. The possibility of conflicts with operational commitments makes the ready-alert periods undesirable for scheduling of other activities.

Another at-home period that is maintained free from training or operational tasking is the forty-five-day period immediately preceding the date a squadron commences a

deployment. This period is normally left open to provide the squadron with a relative quiet period for leave and to make the final preparations for moving to the deployment site. This also allows for an extra period in case a squadron must make up any of the training evolutions that must be accomplished prior to deployment.

B. TRAINING EVENT DESCRIPTION

There are seven primary inspections a squadron must pass prior to a deployment. The training officer is responsible for scheduling these inspections for the squadrons assigned to his wing[1]. For some of the inspections, the order in which they occur relative to other inspections is important, while others may be held anytime. The seven inspections are:

1. NTPI

The Nuclear Training Proficiency Inspection (NTPI) is conducted by a team of inspectors from Commander Nuclear Weapons Training Group Pacific, San Diego, California. It is a records and procedure inspection requiring two uninterrupted working days.

2. Pre-NTPI

The Nuclear Training Proficiency Pre-Inspection (Pre-NTPI) is conducted by personnel from within CPW-10. Its purpose is to ensure the squadron is ready for the NTPI and must be conducted prior to it.

3. CTPI

The Conventional Weapons Technical Proficiency Inspection (CTPI) is conducted by a team from Commander Patrol Wings Pacific, Moffett Field, California (CPWP). In the past this was scheduled approximately three weeks prior to the MRCI. Due to a recent change in policy at CPWP, the CTPI is given with relatively short notice. This prevents scheduling this inspection along with the other inspections.

---

[1]There are four active-duty U.S. Navy land-based patrol air wings. Each have five to seven active duty fixed-wing patrol (VP) squadrons subordinate to them.

4. MRCI

The Mining Readiness Certification Inspection (MRCI) is conducted by an inspection team from the Commanding Officer, Mine Warfare Inspection Group, Charleston, South Carolina. This is a four-working-day-inspection.

5. Pre-MRCI

The Mining Readiness Certification Pre-Inspection (Pre-MRCI) was normally accomplished during the same period as the CTPI. It now takes place approximately three weeks prior to the MRCI. The purpose of this inspection is to ensure the squadron is adequately prepared for the MRCI. Like the Pre-NTPI it is conducted by CPW-10 personnel.

6. CI

The Command Inspection (CI) is an administrative inspection that is performed as the final test that the squadron is ready to go on deployment. This is a one-day inspection performed by CPW-10 personnel as close to and not less than 45 days prior to the deployment start date as possible.

7. NATOPS Inspection

The Naval Air Training and Operating Procedures Standardization (NATOPS) inspection is an aircrew proficiency examination that ensures the squadron's aircrew personnel are knowledgeable of the correct procedures to operate the aircraft safely. This inspection involves written exams for all the aircrew positions in the P-3 Orion aircraft, as well as flight examinations given to three of the flight crews. Ten working days are allowed on the schedule to ensure adequate time to accomplish all the evolutions required for this inspection.

C. THE PROBLEM

Traditional scheduling problems have dealt primarily with activities that are consumers of a given amount of time. The problem has been to optimize the order or placement of the time activities to maximize the efficiency of the total allocated time. Or in other words, find the sequence of events such that there is the least amount of wasted time and the overall output is at a maximum.

The problem faced by the CPW-10 Training Officer is different in that he is not concerned with minimizing the time between events. Actually his intention is to optimize the gap between events such that there is not too much or too little time between events.

An additional subtlety is that conflict arises because the scheduler must not only take into consideration the gaps between events for a specific squadron, but must also consider the interval at which the inspection team is being scheduled.

D. PRIOR APPROACHES

The manual algorithm for arriving at the CPW-10 training schedule has been developed over the years and is informally maintained by the training officers in what is referred to as the Training Officer's turnover notebook. The following algorithm is what has been used in the past and is currently being used by the training officer at CPW-10 to manually develop the yearly training plan [Ref. 7].

1. Fill in the deployment periods for each squadron. A deployment begins and ends on the tenth of a month.
2. Compute the latest date each inspection can take place in accordance with the periodicity requirements. Mark these on the schedule draft.
3. For each squadron, annotate the draft with the following periods:
   a. Its ORE vulnerability period.
   b. Its time between 45 days prior to its deployment and the deployment.
   c. Its post-deployment safety stand-down.
4. Assign the ready-alerts.
   a. The ready-alert periods do not overlap. They begin on either the first or the sixteenth of a month, depending upon whether they are to be a 30-day or 45-day period. 30-day periods are the ideal.
   b. Determine which squadron has the ready-alert the last month of the current planning year. This is the starting point for future ready-alerts.
   c. When selecting a ready-alert do not consider any squadron that:
      (1) Held the ready-alert the month prior to the month being scheduled.
      (2) Is on deployment.
      (3) Is in its post-deployment safety stand-down.
      (4) Is in the ORE vulnerability period.
      (5) Is in the period 45 days prior to the deployment.

d. If more than one squadron remains from part c, select one that has not yet been scheduled for a ready-alert during its current at-home period.

e. If no squadron is available for part d, temporarily skip to the next month and select a squadron in accordance with parts c and d. Then split the skipped month between the previous ready-alert and the selected ready-alert squadron. This will assign the ready-alert to two squadrons over a three-month period.

f. Consider the following as the preferred order to select from available squadrons:

(1) Those in third month or later following post-deployment safety stand-down.

(2) Those in second full month or later after the post-deployment safety stand-down.

(3) Those in first month of ORE vulnerability period.

g. Continue steps c through f until every month has some squadron assigned to a ready-alert.

5. Schedule the NTPI for each squadron (as required):

a. The date must be after the safety stand-down and not during any read alert.

b. Prefer the latest date possible.

c. There must be prior time for the pre-NTPI.

d. Ensure that no major holiday interrupts the pre-NTPI /NTPI block.

6. Schedule each pre-NTPI. The date must fall after the safety stand-down.

7. Using the due dates marked on the draft, schedule the rest of the activities. None should be during a ready-alert and all should be after the post-deployment safety stand-down.

a. The NATOPS evaluation should be as early as practical.

b. The pre-MRCI should be as early in the ORE vulnerability period as practical.

c. The MRCI should be no earlier than two weeks after the pre-MRCI, with same restriction as the pre-NTPI/NTPI.

d. The CI should be on the last working day of the ORE vulnerability period.

8. Make necessary adjustments to optimize the overall schedule and the resources with which it is built.

The final step leaves a great deal of work for the training officer to accomplish. To optimize the schedule requires making judgements on issues that are not always clearly

objective. It is also requires juggling the events in the schedule to make all the events fit. This leaves room for mistakes and, without a clear idea of which is more optimal of two given situations, can lead to inconsistencies in the final schedule. It is at this stage that the experienced scheduler will have a distinct advantage over a newly-assigned individual.

In LCDR Hutson's prototype three separate searches are used to complete the schedule. The first search schedules the ready-alerts. The second search schedules the events that do not occur during the ORE vulnerability period. The third search schedules the events that are during the ORE vulnerability period. With the correct k-factor for the final two searches, this prototype will arrive at a much improved schedule compared to the manually derived schedule. Finding the right k-factor, however, is a matter of trial and error, and the program can then take several days to run. It was also discovered that when converted to M-Prolog, Hutson's program required too much space to run on the computer used for this thesis. [Ref. 7]

IV. PROGRAM DESCRIPTION

A. PROGRAM PERIPHERALS

The program written as part of this thesis was done on an ISI Optimum V Workstation[1] running the UNIX[2] operating system. The language used is M-Prolog because there are both interpreted and compiled versions to run on both the ISIs and IBM-PC[3] compatible machines such as the Zenith-248[4] now common to most military commands. The ISI workstation has the Motorola 68020, 16.67Mhz-chip as its processor.

B. OVERVIEW OF THE NEW APPROACH

In this thesis the scheduling problem was broken down into three separate stages with the intention of greatly improving on the time required by LCDR Hutson's prototype to produce a final schedule. In the first stage the initial database required to build the schedule is established. The second stage divides the year that is to be scheduled into months and determines which month is best for each event to occur in. The third and final stage then optimizes the schedule by determining the best days for each event.

1. New Algorithm

The new algorithm used in this thesis's program is broken into three stages. During each stage a search is conducted.

a. Stage I algorithm

(1) Read the datafile DATABASE.pro and assert its contents as facts in the program's database. The following facts may or may not be contained in the datafile:

(a) Start and finish dates for the schedule

---

[1]"Optimum V Workstation" is a registered trademark of Integrated Solutions Incorporate.

[2]"UNIX" is a registered trademark of ATT.

[3]"IBM-PC" is a registered trademark of the International Business Machines Corporation of Armonk, N.Y., U.S.A.

[4]"Zenith-248" is a registered trademark of Zenith Corporation.

(b) Prior-event dates

(c) Deployment dates

(d) Earmark (or Drop-Dead) dates

(e) Ready-alert schedule

(2) If the start-date, finish-date, prior-event-dates or deployment data were not read in from the datafile, then query the user to get the information.

(3) If the earmark dates were not read in, either query the user or compute them. Use the prior-event dates to compute the earmark dates based on the known qualification durations[1]. The durations are listed in Figure 4.1 [Ref. 8].

Event Name	Qualification Duration
CI	12 months
NATOPS	12 months
NTPI	12 months
MRCI	18 months

Figure 4.1 Event Qualification Lengths

(4) If the ready-alert dates were not read-in, either have the user input them or use a nopath search to determine the optimum arrangement for them. The program used in LCDR Hutson's prototype was used here, modified by adding successor rules to permit scheduling up to three ready alerts per squadron, instead of only two. [Ref. 7].

(5) At the completion of Stage I the program database contains all the facts mention in part (1) above.

b. Stage II algorithm

(1) Generate the **first_pick_months** for each event required to occur during the schedule year, using the table shown in Figure 4.2.

---

[1]A qualification duration begins when a squadron successfully completes the inspection for the qualification. The squadron is required to successfully complete the inspection again prior to the end of the qualification duration.

Inspection	Best Month for Initial Schedule[1]
pre-NTPI	1 month prior to NTPI
NTPI	0 months prior to Earmark date
Pre-MRCI	4 months prior to Deployment start
MRCI	3 months prior to Deployment start
CI	45 days prior to Deployment start
NATOPS	2 months after Deployment finish

Figure 4.2 Event Optimal Month Computation

(2) Using a nopath search, determine the actual months by repeatedly choosing the best schedule and resolving a conflict within that schedule. Resolve a conflict by selecting an event which has a conflict and change its month to vary from its **first_pick_month** by moving it (in order of preference): (1) one month prior, (2) one month later, (3) two months prior, (4) two months later, (5) three months prior, (6) three months later. The new placement cannot conflict with anything else in the schedule. If the selected event cannot be rescheduled without a conflict, then select the next event with a conflict and resolve its conflict. Continue until a schedule results having no conflicts or a minimum number of conflicts. The schedules are evaluated by summing the number of conflicts it contains and the total of its costs as determined using the table shown in Figure 4.4. The best schedule is the one having the lowest evaluation.

(3) The result of Stage II is a month-schedule for the period to be scheduled.

c. Stage III algorithm

(1) Taking the events of the Stage II schedule in chronological order, without back tracking, find the **trialperiod** (sequence of days) for each event during its month that results in a day-schedule with the least total cost (a hill-climbing search).

---

[1]The best month for the initial schedule was determined by studying the actual schedules used during fiscal years 1986 and 1989.

C. MODULE DESCRIPTION

The program was written using the ten modules shown in Figure 4.3. Each of the ten modules is pretranslated (a form of compilation) into separate binary files which are then consolidated into a single binary file that can be run using the M-Prolog interpreter. To run the program in the compiled mode, each module is compiled separately and then consolidated into a single compiled file that can be run.

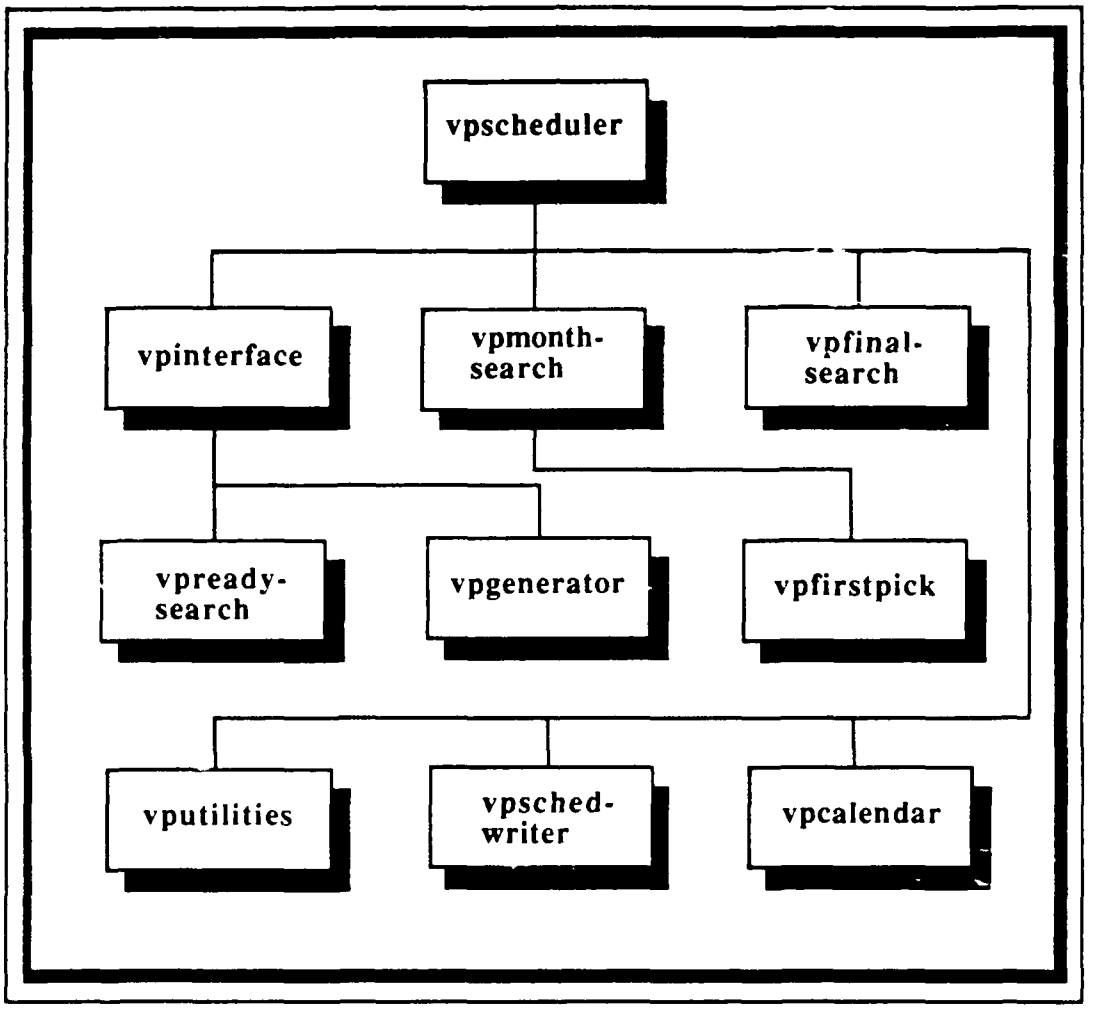

Figure 4.3 Program Module Block Diagram

1. Vpscheduler module

The vpscheduler module contains the top level predicate **go** which is used to call all the other routines in the program. Within this module are routines written by LCDR Hutson for computing and displaying the individual process times as the program runs. It calls the routines to initialize the database, conduct the Stage II search, do the final search, and print the schedule to a file.

2. Vpinterface module

This module contains the routines necessary to initialize the program database. The initialization procedure begins by asking the user if the database file has been updated. If the database file has not been updated, the user is queried for the information necessary to build it.

The database file, DATABASE.pro, is a text file. Each line in the file can easily be read from the file and asserted as a fact in the program database. A sample datafile is shown in Appendix B. Most of the information in this file is stored in the date format[1] to make reading and changing the datafile simpler. An exception to this is the trialperiod[2] information which uses the daynumber format.

If the datafile does not contain all the information required to start Stage II, the interface module will build the datafile. Some of the information must be input by the user, and some of it can be computed. The **yearbegindates**, and **yearenddates**, **prioreventdates**, and deployment data (**prerequisite** facts) must all be contained in the datafile or the user will be queried for the information and the interface module will write it into the datafile.

Following this, all the information contained in the datafile is asserted into the program database as facts. If the datafile did not contain earmark facts, then these will be computed using the prior event dates. The program does not take into account that some

---

[1]The date format is a Prolog list containing the day, the common three-letter abbreviation for the month (using lower-case letters only), and the year. Example: 5 February 1989 in the date format is [5,feb,1989].

[2]A trialperiod is the sequence of days during which an event may occur. A trialperiod is defined by its start daynumber through its finish daynumber.

squadrons may have been given extensions on some of the inspections, delaying the actual earmark date; then the user should put the correct earmark dates in the datafile.

3. Vpreadysearch module

This module is used when the user wants the computer to determine the ready-alert schedule. This is a M-Prolog version of the ready-alert search done by LCDR Hutson, enhanced to schedule up to three ready-alerts periods per squadron. Routines have also been added to convert the readyevents that contain daynumbers[1] as start and finish dates into ready_month facts that contain the month numbers as the start and finish dates, which are used in the Stage II search.

4. Vpgenerator module

The **vpgenerator** module is called to compute the trialperiods if the datafile does not contain them. The final step in initializing the database is to obtain the ready-alert schedule. If the datafile does not contain ready-alert facts, the user has the option of entering the ready-alert events or having the program compute them using the **vpreadysearch** module.This is an M-Prolog version of the generator module written by LCDR Hutson [Ref. 7]. It produces the trialperiods available during a given year fo: each of the different inspections.

5. Vpmonthsearch module

Within this module the Stage II search is conducted. The module begins by finding the first_pick_months[2] for each event that should be scheduled during the year. These assignments form the initial schedule which is passed to a nopath search (see section II.B) which finds the schedule with the least number of conflicts. The total cost of a schedule depends on the number of conflicts it contains and the the amount you must move events from their first_pick_month in order to deconflict it with other events. The table in Figure 4.4 shows the cost of moving an event out of its first_pick_month.

---

[1]The daynumber is the integer representation of a date, equal to the number of days that have occurred between 1 January 1600 and the date being represented.

[2]For each inspection and squadron a first_pick_month can be calculated by using the table shown in Figure 4.2.

(First_pick_month - Event month)	Cost
0	0
1	1
2	2
3	3
-1	0.5
-2	1.5
-3	2.5

Figure 4.4 Cost to Deviate from the First_pick_month

6. Vpfirstpick module

This module is called by the **vpmonthsearch** module to determine which events need to be scheduled during the given year and which month is the preferred or first_pick_month for those events. The first_pick_months are computed using the information in Figure 4.2.

7. Vpfinalsearch module

The final schedule is generated in this module using the ready-alert schedule from Stage I and the month schedule from Stage II. The ready-alert schedule is input as the starting state of a modified hill-climbing search. The search always selects the chronologically earliest unscheduled event from the month schedule. The trialperiods are sequentially tried as possibilities for that selected event. For each, the cost of the schedule including this new event is computed. The cost is then compared with the prior cost computed for a schedule with the same event using the last (an earlier) trialperiod. If the new cost is less than the prior cost, then the new schedule is added to the agenda and the subsequent trialperiod is check for the same event.

The process for selecting the trialperiod for a particular event is concluded when there are no more trialperiods to test or the new cost is greater than the previous cost. The latter condition is due to the convexity of the cost function (see section E). This implies a modified hill-climbing search significantly reduces the search time. The final schedule is

complete when there are no events on the month schedule that have not been added to the final schedule.

8. Vputilities module

This module contains numerous miscellaneous predicates used for list manipulation. Most of this module was provided by Prof. Rowe [Ref. 4].

9. Vpschedwriter module

This module contains the routines used to print a list of events in columnar form. It converts the daynumbers or monthnumbers into common abbreviated dates. Event codes are changed to the more commonly used abbreviations.

10. Vpcalendar module

This is an M-Prolog version of the calendar module written by LCDR Hutson [Ref. 7]. It was enhanced to provide the correct conversion for both date to daynumber and daynumber to date during all years including leap years.

D. DATA STRUCTURES

The majority of the data structures were unchanged from LCDR Hutson's prototype [Ref. 7]. The structures use event codes extensively. The meanings of the event codes are given in Figure 4.5. Some of the structures involving dates use the integer value (daynumber) while others use a Prolog list format such as [1, jan, 1986] for the date January 1, 1986. Because this thesis is additionally concerned with dates in a monthly increment, a unique integer value (monthnumber) is used. Whenever a data structure specifies the need for a date, it is expecting the list format. The daynumber or monthnumber will be given if required.

SYMBOL	EVENT NAME
ec	Command Inspection
en	NATOPS Inspection
ewc	pre-MRCI/CTPI
ewm	MRCI
ewp	pre-NTPI
ewn	NTPI
tr1a	first ready-alert
tr1b	second ready-alert
tr1c	third ready-alert
dv	ORE vulnerability period
ds	Post-deployment Safety-Standown
dr	Operational Deployment

Figure 4 .5 Event Symbol Definitions

Most of the structures that involve an event are facts in the program database. These facts have a predicate name defining what type of event it is and either three or four arguments. The first argument is the squadron. The second argument is the symbol for the inspection or event. The third and fourth arguments are daynumbers or dates. An example is:

priorevent(vp19,tr1a,140963,140994).

This is a fact that occurred prior to the start of the period being scheduled. The squadron,VP-19, had its first ready-alert starting on the date associated with the daynumber 140963 and ending on the date associated with daynumber 140994.

The following are similar structures that were carried over from LCDR Hutson's prototype [Ref. 7]:

prerequisite(<squadron>, <event code>, <start daynumber>, <finish daynumber>)

priorevent(<squadron>,<event code>, <start daynumber>, <finish daynumber>)

earmark(<squadron>,<event code>, <date>)

trialperiod(<event code>, <start daynumber>, <finish daynumber>)

readyevent(<squadron>, <event code>, <start daynumber>, <finish daynumber>)

The following new structures were created for this thesis:

ready_month(<squadron>, <monthnumber>)

ready_half_month(<squadron>, <monthnumber>)

prioreventdate(<squadron>,<event code>, <start date>, <finish date>)

prerequisitedate(<squadron>,<event code>, <start date>, <finish date>)

During stage III an indexing scheme is used in the data structures for the search. Each state or list of events that make a schedule is given a unique integer value. Rather than passing the list of events from rule to rule during the search, only the index is passed. The indexing is used to associate the squadrons costs and the team events costs to the particular state.

E. COST FUNCTIONS

In this thesis three cost functions were developed. The optimum interval between a pre-inspection and the actual inspection is three weeks, not one month [Ref. 10]. It is also less critical for an inspection team to be scheduled every thirty days than it is for a squadron. The three cost functions, therefore are broken into the cases for:

(a) squadron related-events cost

(b) squadron unrelated-events cost

(c) team cost.

Prof. Rowe assisted in the development of the polynomial functions used in the three cost functions. Each has a single minimum value of 1.0. The major differences among the functions are the values at which the minimum occurs and the rate of increase for delay periods greater than where the minimum occurs. In each case, the cost of a delay that is C less than the optimum value is greater than the cost for a delay amount C greater than the optimum value. The squadron cost for delays greater than 180 days is made zero because deployments are normally six months in duration. There should not be a cost associated

with the delay measured between the last event preceding a deployment and the first event following it.

The squadron related-events cost formulas given in Figure 4.6 were developed to have a minimum cost occur at a delay of twenty-one days, a cost value of one hundred at a delay of zero days, and a cost of ten at a delay of forty-two days.

Delay (D)	Cost Formula
< 21	Cost = 100 - 243 * (D/21) + 189 * (D/21)^2 - 45 * (D/21)^3
>= 21	Cost = 10 - 6/7 * D + 1/49 * D^2
>180	Cost = 0

Figure 4.6 Squadron Related-events Cost Formulas

The squadron unrelated-events cost formulas given in Figure 4.7 were developed to have a minimum cost occur at a delay of thirty days, a cost value of one hundred at a delay of zero days, and a cost of ten at a delay of sixty days.

Delay (D)	Cost Formula
< 30	Cost = 100 - 243 * (D/30) + 189 * (D/30)^2 - 45 * (D/30)^3
>= 30	Cost = 10 - 0.6 * D + 0.01 * D^2
>180	Cost = 0

Figure 4.7 Squadron Unrelated-events Cost Formulas

The team-event cost formulas, given in Figure 4.8, were developed to have a minimum cost occur at a delay of thirty days, a cost value of one hundred at a delay of zero days, and a cost of ten at a delay of sixty days with small increases in cost thereafter.

Delay (D)	Cost Formula
< 51	Cost = 100 - 247.25 * (D/30) + 197.5 * (D/30) ^2 - 49.25 * (D/30)^3
>= 51	Cost = 8.52 + 1/(50/D)^2

Figure 4.8 Team-events Cost Formulas

Each of the three cost have an identical evaluation function, the number of unscheduled events, which is used to determine when the search has reached its goal.

V. PROGRAM RESULTS

A. TEST DATA

The data collected and reported in [Ref. 7] was used for comparing the results of LCDR Hutson's program [Ref. 7], the manual schedule, and this program. This data was used to create the CPW-10 Training schedule containing upto six events for each of the seven squadrons during fiscal year 1986. In order to more thoroughly test this program, additional data was collected from the CPW-10 Training office personnel [Ref. 8,9,10], covering the fiscal year 1989. A working version of LCDR Hutson's prototype was not available to allow further comparisons using the new data.

B. STORAGE REQUIREMENTS

When all the program was consolidated into a single binary module that runs using the M-Prolog interpreter, it required 124928 bytes of storage. The consolidated compiled program program required 167936 bytes of storage. Depending on the year being scheduled, the main stack size varied from 3496 using 1986 data to only 2827 using 1989 data. Likewise the statement table varied from 74967 to 63340, using 1986 and 1989 data respectively.

C. STAGE I TIMES

The cpu time required to run Stage I of the program is dependent upon the amount of information that is preloaded into the datafile. When the user knows the ready-alert schedule, the earmark dates, and the priorevent dates, and the trialperiod information has all been computed from prior runs, then Stage I is a matter of the program reading the information from the datafile and asserting it into the program database as Prolog facts, plus adding readymonth facts. When done this way, Stage I required approximately 24 to 28 cpu seconds.

If the datafile contains the minimal information of only the priorevents and the deployment-related dates, then Stage I required 140 to 182 cpu seconds to complete. The process of computing the trialperiods for the schedule year, asserting them as facts in the

program database and writing them to the database file required an average of 0.165 cpu seconds per trialperiod. The time required to compute the earmark dates, assert them as facts in the database, and write them to the database file was approximately 2 to 3 cpu seconds. For the computer to calculate the earmark dates, the datafile must contain all the priorevents completed, not just those completed since the squadrons' last deployment.

The final step of Stage I, the search for the ready-alert schedule varied a great deal depending on the year. The average time for 1986 was 6 cpu seconds, and for 1989, 27 cpu seconds.

The diagram in Figure 5.1 shows the total times required for Stage I depending on the datafile content. The time required to produce a schedule varies depending on the year and the number of events to be scheduled. Every fiscal year will not have the same number of trialperiods available due to the placement of holidays which must be scheduled around. Depending on the deployment cycles, the number of inspections required during different fiscal years will also vary.

Stage I Times

Priorevents Known?	Deployment Data Known?	Earmark Dates Known?	Trialperiods Known?	Ready Alerts Assigned?	CPU Time 1986	CPU Time 1989
YES	YES	YES	YES	YES	24.08	27.18
YES	YES	YES	YES	NO	27.74	51.28
YES	YES	YES	NO	NO	143.74	171.66
YES	YES	NO	NO	NO	139.74	181.74

YES - indicates the information was pre-loaded into the datafile

time - is in cpu seconds

Figure 5.1 Stage I Times

D. STAGE II TIMES

The times for Stage II were dependent on the number of conflicts existing in the initial schedule made from the first_pick_month. We also show the number of conflicts that could not be resolved by attempting to reschedule conflicting events within three months either side of their first_pick_month. Below is a diagram showing the times for Stage II for different schedule years.

Stage II Times

Fiscal Year	Events to be Scheduled	Initial Conflicts	Unresolved Conflicts	Stage II Time	Average Time / Event
1986	31	6	0	220.0	7.10
1989	22	6	0	48.0	2.18
time - is in cpu seconds					

Figure 5.2 Stage II Times

E. STAGE III TIMES

The final search (Stage III) was the largest consumer of time. Two techniques discovered useful to reduce the time for Stage III were indexing the cost and determining when the minimum cost was reached. When the scheduler program was run with Stage III doing a normal hill-climbing search, the time for Stage III was 1408 cpu seconds for the 1986 schedule and 712 cpu seconds for the 1989 schedule.

Finding the minimum cost for adding a new successor was possible because the trialperiods for events being added were generated sequentially beginning at the first period of the month, and because the events were added to the schedule beginning with those in the schedule-start-month chronologically progressing through the year. Since all cost functions used are convex, the minimum cost is when the total cost is no longer decreasing. When the hill-climbing search was so modified, the Stage III time was reduced to 840 cpu seconds for the 1986 and to 493 cpu seconds for the 1989 schedules.

Using a hill-climbing search, each time a new successor is added to the current schedule, the cost of the new schedule must be found. The function to compute the cost of a schedule is done in two parts, the squadron cost and the inspection team cost, as explained in Hutson's thesis [Ref. 7]. Without indexing all the individual squadron-costs

and all the team-costs are recomputed for each new schedule. With indexing only the squadron-cost for the new event's squadron and the team-cost of the inspection team associated with the new event is recomputed. The remaining costs, referenced by indexes, are brought forward from the previous schedule and must only be added to arrive at the new total cost. With the costs indexed, so that only the new cost for the specific squadron and the new cost for the specific inspection being added to the schedule were computed as each new successor was added, the time for Stage III was further reduced to 308 cpu seconds for 1986 and to 196 cpu seconds for 1989.

FY	Time Using Original Search	Time Using Min Cost Search	Reduction	Time Using Min Cost & Indexing	Reduction
1986	1408	840	40%	308	78%
1989	712	494	31%	196	72%

Figure 5.3 Summary of Stage III Time Improvements

F. SCHEDULE GENERAL QUALITY

Though the schedules produced using this program have not achieved the absolute optimized results, the trade-off of reduced computation time for optimization is more than favorable. The average number of days between events for a squadron for the 1986 schedule is 21.85 days. The average number of days between events for an inspection team is 61.32. Five days is the smallest delay between two consecutive events for a single squadron. Three days is the smallest delay between two consecutive events for an inspection team. The schedules for both 1986 and 1989 are reasonable.

G. IMPROVEMENTS MADE OVER LCDR HUTSON'S PROGRAM

The objective of this thesis's approach to the scheduling problem was to reduce the search space to enable the searches for the optimum schedule to be completed in a shorter period of time on a less capable computer system. Comparing the total of 22534 cpu

seconds required to produce the 1986 schedule using LCDR Hutson's program and only 555 cpu seconds using this thesis's program, it is obvious we did achieve the objective. The idea of starting with first_pick_months in Stage II came from studying the manual algorithm and prior training schedules that were developed using it. Prior schedules showed there is a high correlation to the order of the events for a squadron and its deployment cycle. By rescheduling events in conflict, the result of Stage II is a schedule with a better spread of the events.

The search in Stage III is a hill-climbing search and is the principle reason for the overall speed increase over the Hutson prototype. Since the ready-alert schedule is the start state for this search, the decision of where a new event will best fit into the schedule will take into consideration the prior-events and any ready-alert for the same squadron. This causes the program to position events closer to the center of a month if the squadron has ready-alerts the month prior and the month following.

The interface module provides improvements to LCDR Hutson's program by allowing flexibility in the initial database set-up. The user can have the computer determine the ready-alerts or manually input the schedule. Once the program has run on a given schedule year, the database file will contain trialperiods which can be read into the program during future runs on the same schedule year, avoiding the time-consuming process of recomputing these trialperiods.

The information in the Database.pro file is self-explanatory and can be modified directly by anyone familiar with using a text editor program. Because the initial data is read in from the Database.pro file, making improvements to the user interface is easy. Any interface can be used that will produce the Database.pro file in the same format. An example would be to use a HyperCard[1] program to build the file.

---

[1]"HyperCard" is a register trademark of Apple Computer, Incorporate, Cupertino, California.

VI. CONCLUSIONS

This thesis has demonstrated that a microcomputer does have the power to perform the computations necessary to produce a schedule much quicker than can be done by hand. The schedule produced using this thesis's program is also more optimal than the manual schedule. It was not quite able to achieve the same degree of optimization achieved by the Hutson prototype, but was much faster. The degree of optimization achieved is illustrated in Figure 6.1, which shows the total costs calculated using the same cost function on the 1986 schedules produced using the manual method, the Hutson prototype, and this thesis's method.

Scheduling Method	Total Cost Calculated
Manual	831.6
Hutson's Prototype	596.7
This Program	675.7

Figure 6.1 Total Cost for 1986 Schedule

This thesis also shows the significant time savings of indexing the cost in each state of the hill-climbing search over computing all the cost each time a new successor is found. But if simpler cost functions are used this technique might not result in a significant savings of time.

This program is ready for use at CPW-10, once the code is converted to run on IBM-compatible computers. Changing the squadron names that have been hardcoded into the program is all that is necessary to make the program adaptable for the other three Patrol Wings.

The user interface with this program can use a great deal of improvement. Any interface that provides a way of updating the database file could be used. An interface that could quickly and easily change the database file would greatly enhance this program. The

results of the program are currently hardcoded to be written to the file SCHEDULE.pro. An interface that allows the user to change the output file would also be helpful.

This thesis did not research the cost function used in the Stage III search. Thirty days between events may not actually be the ideal. This is an area for further thesis work.

APPENDIX A - PROGRAM SOURCE CODE

VPSCHEDULER MODULE

This is the module that initiates the scheduling program.

```
module main.

import ( go /0).

/*$eject*/
body.

:- go.

endmod /* main */ .

module vpscheduler.

export  ( go /0, successor_counter /0, give_count /2, processtime /2, display_statistics /0).

import  ( generate_first_pick / 0,generate_final_schedule / 3, search1 / 1, ready_month /2,
      search2 /1, union / 3, append /3, create_datafile / 0, read_datafile / 0,
      convert_date_data / 0, complete_database / 1, nopathsearch/ 2, open / 2, tell / 2,
      told / 1, database_conversion /0, get_trialperiods /0, sortevent /2, depthsearch /2,
      prettyprint /1, myname /2, stars /0,state / 2,  daynumber_to_date /2, yearend /1,
      squadronlist /1,  eventprinter / 1, count1 /1 ,count2 /2 ,count3 /1).

global (status,trialperiod,prerequisite,event,search1,
       jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,
       monday,tuesday,wednesday,thursday,friday,saturday,sunday,weekend,
        tr1a,tr1b,tr1c,ewp.ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
       vp9,vp19,vp40,vp46,vp47,vp48,vp50).

/*$eject*/
body.

dynamic(other_process_time/1).
dynamic(sub_successorcount/1).
dynamic(total_successorcount/1).
```

```
go:-
      start_time,
      display_statistics,
      init_dynamics,
      create_datafile,
      read_datafile,
      convert_date_data,
      processtime("converting the datafile",_),
      complete_database(Answer1),
      processtime("completing the database (Stage I)",_),
      write("The total time to complete Stage I is :"),nl,
      display_statistics,
      search2(Answer2),!,
      eventprinter(Answer2),
      processtime("generating monthly schedule (Stage II)",_),
      generate_final_schedule(Answer1,Answer2,Answer3),
      sortevent(Answer3,Answer4),
      eventprinter(Answer4),
      processtime("generating final schedule (Stage III)",_),
      write("WRITING THE FINAL SCHEDULE TO THE FILE
          $"SCHEDULE.pro$"..."),  nl,
      open(1,"SCHEDULE.pro"),
      tell(1,"SCHEDULE.pro"),
      write("THE FOLLOWING IS THE COMPUTED SCHEDULE: "),nl,nl,
      eventprinter(Answer4),
      told(1),
      display_statistics,
      write("The Scheduler is now FINISHED..."),nl,

go:-
      write("UNABLE TO DETERMINE SCHEDULE. VERIFY DATABASE INPUT
          $AND TRY AGAIN."),nl,!.

/* change system date to common format. 47 is '/' ; 48 is '0' .        */

convert_date(SystemDate,Date):-
      myname(SystemDate,LIST),
      LIST = [A,B,C,D,E,F],
      myname(Date,[A,B,47,C,D,47,E,F]).

convert_date(SystemDate,Date):-
      myname(SystemDate,LIST),
      LIST = [B,C,D,E,F],
      myname(Date,[48,B,47,C,D,47,E,F]).

/* change system time to common format. 58 is ':' ; 48 is '0'.         */

convert_time(SystemTime,Time):-
      myname(SystemTime,LIST),
      LIST = [A,B,C,D,E,F],
      myname(Time,[A,B,58,C,D,58,E,F]).
```

```
convert_time(SystemTime,Time):-
      myname(SystemTime,LIST),
       LIST = [B,C,D,E,F],
       myname(Time,[48,B,58,C,D,58,E,F]).

start_time:-
      state(system_date,SYSDate),
      convert_date(SYSDate,DATE),
      write(DATE),write_spaces(1),
      state(system_time,SYSTime),
      convert_time(SYSTime,TIME),
      write(TIME),nl,nl,
       write("THE SCHEDULER IS PROCESSING ..."),nl,nl,!.

give_count(Process,Time):-
       sub_successorcount(SSK),
      del_all_statements(sub_successorcount/1),
      add_statement(sub_successorcount(0)),
      write("==> Successors generated during "),write(Process),write(":"),
       write(SSK),nl,
      Average is Time/SSK,
      write("==> Average process time for successors in "),
       write(Process),write(":"),write(Average),write(" cpu seconds."),nl,!.

final_count:-
      total_successorcount(TSK),
      state(cpu_time,X),
      Overall is X/(TSK*1000),
      write("Total successors = "),write(TSK),
      write("and overall Average Processing Time = "),write(Overall),
       write(" cpu seconds."),nl,!.

successor_counter:-
       sub_successorcount(SSK),
      total_successorcount(TSK),
      del_statement(sub_successorcount(SSK)),
      del_statement(total_successorcount(TSK)),
       SSK2 is SSK + 1,
       TSK2 is TSK + 1,
      add_statement(sub_successorcount(SSK2)),
      add_statement(total_successorcount(TSK2)),!.

processtime(Process,FT):-
      state(cpu_time,X),
      other_process_time(OPT),
      del_statement(other_process_time(OPT)),
      FT is (X - OPT)/1000,
      write("The process for "),write(Process),
      write(" completed in: "),write(FT),write(" cpu seconds."),nl,
      add_statement(other_process_time(X)),!.
```

```
display_statistics:-
        stars,stars,
        write_tab(18),write("SYSTEM STATUS"),nl,
        state(cpu_time,X),
        write("cpu time = "),write(X),write(" msec"),nl,
        state(main_stack,[U,C]),
        write("main stack used = "),write(U),nl,
        state(statement_table,[U1,C1]),
        write("statement table used = "),write(U1),nl,

init_dynamics :-
        add_statement(other_process_time(0)),
        add_statement(sub_successorcount(0)),
        add_statement(total_successorcount(0)),
        add_statement(count1(0)),
        add_statement(count2(0)),
        add_statement(count3(0)).

endmod  /* module vpscheduler */ .
```

VPINTERFACE MODULE

This module reads the database file DATABASE.pro and asserts the appropriate facts into the program database.

```
module vpinterface .

export (yearbegindate / 1,  yearenddate / 1, create_datafile / 0, read_datafile / 0,
     convert_date_data / 0, complete_database / 1, earmark / 3, eventcode / 2,
     eventnames / 1, squadronlist / 1, priorevent / 4, prerequisite / 4, earmark / 3,
     eventcode / 2, teamevents / 1, yearbegin / 1, yearend / 1).

import (member / 2, myname / 2, search1 / 1, datetodaynumber / 2,
     daynumber_to_date / 2, trialperiod / 3, display_statistics / 0, month_to_number /3,
     number_to_month /3,generate_trialperiods /0,  build_ready_months / 1,
     readyevent / 4, build_readyevents / 1, eventprinter /1, open / 2, tell / 2, told / 1 ).

global (readyevent,priorevent,prerequisite,earmark,trialperiod,
     status,prerequisite, yearbegindate, yearenddate,event,
     prioreventdate,prerequisitedate,readyeventdate,
     ntpi,pre_mrci,mrci,natops,command,ready_alert,pre_ntpi,
      jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,
      monday,tuesday,wednesday,thursday,friday,saturday,sunday,weekend,
       tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
       vp9,vp19,vp40,vp46,vp47,vp48,vp50,y,n,quit).

/*$eject*/
body.

dynamic(yearbegindate/1).
dynamic(yearenddate/1).
dynamic(yearbegin/1).
dynamic(yearend/1).
dynamic(priorevent/4).
dynamic(prerequisite/4).
dynamic(earmark/3).
dynamic(prioreventdate/4).
dynamic(prerequisitedate/4).
dynamic(readyeventdate/4).

create_datafile :-
     newpage,
     datafileinfo,
     write("Has the file $"DATABASE.pro$" been updated? (Y/N)"),nl,
     get_answer(Ans),nl,nl,
     handle_answer(Ans,ReadySched),!.

newpage :- nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl,nl.
```

```
datafileinfo :-
     write("THIS PROGRAM WILL READ INFORMATION FROM A DATA FILE
          $NAMED "), nl,
     write("$"DATABASE.pro$" OR "),nl,
     write("IT WILL ASK YOU TO TYPE-IN THE REQUIRED INFORMATION TO "),
     nl, write("BUILD A NEW DATA FILE NAMED $"DATABASE.pro$" ."), nl, nl,
     write("PLEASE NOTE: BUILDING A NEW DATA FILE WILL DESTROY THE "),
     nl, write("INFORMATION IN THE OLD DATA FILE"),nl,nl,nl,nl,nl.

handle_answer(ANS,ReadySched) :-
     affirm_answer(ANS),!.    /* data file is current */

handle_answer(ANS,ReadySched) :-
     neg_answer(ANS),         /* data file needs updated */
     double_check,
     update_database_file,!.

double_check :-
     newpage,
     write("CONTINUING WILL DESTROY THE CURRENT CONTENTS OF THE
          $FILE"), nl,
     write("$"DATABASE.pro$"....ARE YOU SURE THAT YOU WANT TO
          $CONTINUE ?"),
     write(" (Y/N) "), nl, nl, nl,
     get_answer(ANS),
     handle_double_check(ANS).

handle_double_check(ANS) :-
     affirm_answer(ANS).

handle_double_check(ANS) :-
     halt.

update_database_file:-
     open(3,"DATABASE.pro"),
     update_scheduledates,
     update_priorevents,
     update_deployments,
     told(3).

update_scheduledates :-
     write("Enter the schedule START date -- (ex. 1 oct 1985)"),nl,
     get_date(YSD,YSM,YSY),nl,nl,
     write("Enter the schedule FINISH date -- (ex. 30 sep 1986)"),nl,
     get_date(YFD,YFM,YFY),nl,nl,
     tell(3,"DATABASE.pro"),
     write("yearbegindate(["),write(YSD),write(","),
     write(YSM),write(","),write(YSY),write("])"),nl,
     write("yearenddate(["),write(YFD),write(","),
     write(YFM),write(","),write(YFY),write("])"),nl,
     told(3).
```

```
update_priorevents :-
      priorevent_header,
       member(SQ,[9,19,40,46,47,48,50]),
      write("Squadron: VP-"),write(SQ),nl,nl,
      member(Event,[tr1a,en,ec,ewp,ewn,ewc,ewm]),
      add_priorevents(Event,SQ,3,"DATABASE.pro"),     /* 3 is output channel */
      fail.

update_priorevents :- !.

priorevent_header :-
      write("Enter the most recent finish dates for each of the"), nl,
      write("events/inspections conducted for each squadron."),nl,nl,
      dashes.

add_priorevents(EVT,SQ,CH,OUTFILENAME) :-
      switch(EVT,EVTNAME),
      write("Last "),
      write(EVTNAME),
      write(" END DATE: (31 jul 1986)"),nl,
      get_date(DAY2,MO2,YR2),nl,
      DAY1 is DAY2 - 1,              /* the start date of the priorevents is not important */
      write_data("prioreventdate",CH,OUTFILENAME,SQ,EVT,DAY1,MO2,YR2,DAY2,
           MO2,YR2),!.

switch(tr1a,"Ready Alert").
switch(en,"NATOPS").
switch(ec,"Command Inspection").
switch(ewp,"pre-NTPI").
switch(ewn,"NTPI").
switch(ewc,"PRE_MRCI").
switch(ewm,"MRCI").

update_deployments :-
      deploymentheader,
      member(SQ,[9,19,40,46,47,48,50]),
      get_ore_dates("prerequisitedate",SQ,3,"DATABASE.pro"),
      get_ssd_dates("prerequisitedate",SQ,3,"DATABASE.pro"),
      get_deployment_dates("prerequisitedate",SQ,3,"DATABASE.pro"),
      fail.

update_deployments :- !.

get_ore_dates(PRED,SQ,CH,OUTFILENAME) :-
      write("Squadron: VP-"),write(SQ),nl,nl,
      write("ORE PERIOD START DATE: (ex. 01 jul 1986)"),nl,
      get_date(DAY1,MO1,YR1),nl,
      write("ORE PERIOD END DATE: (30 sep 1986)"),nl,
      get_date(DAY2,MO2,YR2),nl,
      write_data(PRED,CH,OUTFILENAME,SQ,dv1,DAY1,MO1,YR1,DAY2,MO2,
           YR2), !.
```

```
get_ssd_dates(PRED,SQ,CH,OUTFILENAME) :-
      write("Squadron: VP-"),write(SQ),nl,nl,
      write("Enter information for the SAFETY STAND-DOWN period PRIOR to"),nl,
      write("the above ORE PERIOD"),nl,nl,
      write("SAFETY STAND-DOWN PERIOD START DATE: (ex. 11 dec 1985)"),nl,
      get_date(DAY1,MO1,YR1),nl,
      write("SAFETY STAND-DOWN PERIOD END DATE: (ex. 10 jan 1986)"),nl,
      get_date(DAY2,MO2,YR2), nl,
      write_data(PRED,CH,OUTFILENAME,SQ,ds0,DAY1,MO1,YR1,DAY2,MO2,
          YR2),!.

get_deployment_dates(PRED,SQ,CH,OUTFILENAME) :-
      write("Squadron: VP-"),write(SQ),nl,nl,
      write("Enter information for the DEPLOYMENT period FOLLOWING "),nl,
      write("the above ORE PERIOD"),nl,nl,
      write("DEPLOYMENT PERIOD START DATE: (ex. 16 nov 1986)"),nl,
      get_date(DAY1,MO1,YR1),nl,
      write("DEPLOYMENT PERIOD END DATE: (ex. 05 may 1987)"),nl,
      get_date(DAY2,MO2,YR2), nl,
      write_data(PRED,CH,OUTFILENAME,SQ,dr1,DAY1,MO1,YR1,DAY2,MO2,
          YR2),
      compute_ssd2(SQ,DAY2,MO2,YR2,PRED,CH,OUTFILENAME),!.

 /* Adds the safety standdown following the deployment */
compute_ssd2(SQ,DAY,MO,YR,PRED,CH,OUTFILENAME) :-
      datetodaynumber([DAY,MO,YR],DeploymentFDN),
      SDN is DeploymentFDN + 1,
      FDN is SDN + 30,
      daynumber_to_date(SDN,[DAY1,MO1,YR1]),
      daynumber_to_date(FDN,[DAY2,MO2,YR2]),
      write_data(PRED,CH,OUTFILENAME,SQ,ds1,DAY1,MO1,YR1,DAY2,MO2,
          YR2),!.

read_datafile :-
      newpage,
      write("READING DATA FROM THE FILE $"DATABASE.pro$".....),nl,
      set_channel(infile(2),name = "DATABASE.pro"),
      set_input(infile(2)),
      read_datafile2,
      fail.

read_datafile :-
      close_input(infile(2)).

read_datafile2 :-
      read_token(file end),nl,nl, + .
```

```
read_datafile2 :-
      read(X),
      write("."),
      add_statement(X),
      read_datafile2.

convert_date_data :-
      prioreventdate(SQ,E,S,F),
      datetodaynumber(S,SDN),
      datetodaynumber(F,FDN),
      add_statement(priorevent(SQ,E,SDN,FDN)),
      fail.

convert_date_data :-
      prerequisitedate(SQ,E,S,F),
      datetodaynumber(S,SDN),
      datetodaynumber(F,FDN),
      add_statement(prerequisite(SQ,E,SDN,FDN)),
      fail.

convert_date_data :-
      readyeventdate(SQ,E,S,F),
      datetodaynumber(S,SDN),
      datetodaynumber(F,FDN),
      add_statement(readyevent(SQ,E,SDN,FDN)),
      fail.

convert_date_data :-
      del_all_statements(prioreventdate/4),
      del_all_statements(prerequisitedate/4),
      del_all_statements(readyeventdate/4).

complete_database(READYLIST) :-
      year_to_schedule,
      write("Updating the earmark events...."),nl,
      display_statistics,
      update_earmarks,
      write("Finished updating the earmark events ..."),nl,
      display_statistics,
      delete_old_priorevents,   /* deletes the priorevents before ds0 */
      write("Updating the trialperiod events...."),nl,
      display_statistics,
      update_trialperiods,
      write("Finished updating the trialperiod events ..."),nl,
      display_statistics,
      write("Updating the readyalert events...."),nl,
      display_statistics,
      update_readyalerts(READYLIST),
      write("Finished updating the readyalert events ..."),nl,
      display_statistics.
```

```
update_readyalerts(READYLIST) :-
      readyevent(SQ,EVT,SDN,FDN), /* the readyevents were in datafile */
      newpage,
      write("THE READY ALERT SCHEDULE CONTAINED IN THE DATA FILE"),nl,
      write("IS NOW BEING READ INTO THE PROGRAM...."),nl,nl,
      make_readylist([],READYLIST),
      build_ready_months(READYLIST),
      write("THE READY ALEART SCHEDULE FOLLOWS:"),nl,
      dashes,nl,
      eventprinter(READYLIST).

update_readyalerts(READYLIST) :-
      newpage,
      write("Do you want the computer to determine the Ready Alert "),
      write("schedule ? (Y/N)"),nl,nl,nl,nl,
      get_answer(ANS),
      process_ready_answer(ANS,READYLIST),
      add_readyevents_to_datafile(READYLIST).

process_ready_answer(ANS,READYLIST) :-
      affirm_answer(ANS),      /* let the computer figure R/A sched */
      search1(READYLIST).

process_ready_answer(ANS,READYLIST) :-
      neg_answer(ANS),       /* ask user for R/A sched */
      get_readylist(READYLIST),
      build_readyevents(READYLIST),
      build_ready_months(READYLIST).

get_readylist([]) :-
      get_squadron(SQ),
      SQ = quit.

get_readylist([event(SQ,EVT,SD,FD)|RList]) :-
      get_squadron(SQ),
      get_ready_daynumbers(SDN,FDN),
      get_event(EVT),
      get_readylist([RList]) .

get_squadron(SQ):-
      write("Enter the squadron -- (ex. vp9) "),nl,
      write(" or type $"quit$" when finished.   "),nl,
      read_record(SQ),
      valid_squadron(SQ).

get_squadron(SQ):-
      read_record(SQ),
      write(SQ),
      write("is not valid input -- try again"),
      get_squadron(SQ).
```

```
valid_squadron(vp9).
valid_squadron(vp19).
valid_squadron(vp40).
valid_squadron(vp46).
valid_squadron(vp47).
valid_squadron(vp48).
valid_squadron(vp50).
valid_squadron(quit).

get_ready_daynumbers(SDN,FDN) :-
      write("Enter the START date -- (ex. 1 apr 1986)"),nl,
      get_date(D1,M1,Y1),nl,
      datetodaynumber(SDN,[D1,M1,Y1]),
      write("Enter the FINISH date -- (ex. 30 apr 1986)"),nl,
      get_date(D2,M2,Y2),nl,
      datetodaynumber(FDN,[D2,M2,Y2]).

get_event(EVT) :-
      write("Enter the number of Ready Alerts this will make for this"),nl,
      write(" squadron during this schedule year -- "),nl,
      write("(ie. enter 1 for the first, 2 for the second, etc)."),nl,
      read_record(STR),
      convert(STR,string,NUM),
      valid_num(NUM),
      convert_num(NUM,EVT).

get_event(EVT) :-
      read_record(STR),
      write(STR),
      write(" was not a valid entry -- try again."),nl,
      get_event(EVT).

valid_num(1).
valid_num(2).
valid_num(3).

convert_num(1,tr1a).
convert_num(1,tr1b).
convert_num(1,tr1c).

make_readylist(L1,L2) :-
      readyevent(SQ,E,S,F),
      not(member(event(SQ,E,S,F),L1)),
      make_readylist([event(SQ,E,S,F)|L1],L2).

make_readylist(L1,L1) .
add_readyevents_to_datafile([]) .
```

```
add_readyevents_to_datafile([event(SQ,E,SDN,FDN)|READYLIST]) :-
      tell(3,"DATABASE.pro"),
      write("readyeventdate("),write(SQ),write(","),
      write(E),write(","),
      daynumber_to_date(SDN,SD),
      daynumber_to_date(FDN,FD),
      write(SD),write(","),write(FD),write(")"),
      told(3),
      add_readyevents_to_datafile(READYLIST).

update_earmarks :-
      earmark(SQ,EVT,FDN).   /* the earmarks are current */

update_earmarks :-
      newpage,
      write("COMPUTING THE EARMARK DATES FOR EACH
      SQUADRON...."),nl,nl,nl,nl,
      priorevent(SQ,EVT,SDN,FDN),
      daynumber_to_date(FDN,FD),
      compute_earmark(EVT,FD,[DAY1,MO1,YR1]),
      sqd_to_number(SQ,SQN),
      write_data(earmark,3,"DATABASE.pro",SQN,EVT,DAY1,MO1,YR1),
      add_statement(earmark(SQ,EVT,[DAY1,MO1,YR1])),
      fail.

update_earmarks :- !.

compute_earmark(Insp,[FD,FMO,FYR],[FD,EMO,EYR]) :-
      periodicity(Insp,Period),
      month_to_number(FMO,FYR,MN),
      EMN is MN + Period,
      number_to_month(EMO,EYR,EMN),!.

/* The periodicity is the number of months an inspection is good for.        */

periodicity(ec,12).
periodicity(en,12).
periodicity(ewc,18).
periodicity(ewm,18).
periodicity(ewn,12).

/*     remove the priorevents done before the last safety standdown          */

delete_old_priorevents :-
      eventnames(EVTList),
      squadronlist(SQList),
      member(SQ,SQList),
      prerequisite(SQ,ds0,SDN,FDN),
      member(EVT,EVTList),
      priorevent(SQ,EVT,SDN1,FDN1),
      FDN1 < SDN,
      del_statement(priorevent(SQ,EVT,SDN1,FDN1)).
```

```
delete_old_priorevents :- !.

update_trialperiods :-
      trialperiod(EVT,SDN,FDN).   /* the database is current */

update_trialperiods :-
      newpage,
      generate_trialperiods,
      store_in_datafile.

store_in_datafile :-
      tell(3,"DATABASE.pro"),
      trialperiod(EVT,S,F),
      write("trialperiod("),write(EVT),write(","),write(S),write(","),
      write(F),write(")"), nl,
      fail.

store_in_datafile :-
      told(3).

write_data(PRED,CH,OUTFILENAME,SQ,EVT,D1,M1,Y1,D2,M2,Y2) :-
      tell(CH,OUTFILENAME),
      write(PRED),write("("),
      write("vp"),write(SQ),write(","),
      write(EVT), write(",["),
      write(D1),write(","),
      write(M1),write(","),
      write(Y1),write("],["),
      write(D2),write(","),
      write(M2),write(","),
      write(Y2),write("])"),
      told(CH), !.

write_data(PRED,CH,OUTFILENAME,SQ,EVT,DAY1,MO1,YR1) :-
      tell(CH,OUTFILENAME),
      write(PRED),write("("),
      write("vp"),write(SQ),write(","),
      write(EVT), write(",["),
      write(DAY1),write(","),
      write(MO1),write(","),
      write(YR1),write("])"),
      told(CH),!.

get_date(DAY,MO,YR) :-
      read_record(DATE),
      myname(DATE,DATELIST),
      convertdate(DATELIST,DAY,MO,YR),!.
```

```
convertdate([D1,D2,SP,M1,M2,M3,SP,Y1,Y2,Y3,Y4],DAY,MO,YR) :-
      myname(DAYS,[D1,D2]),
      convert(DAYS,number,DAY),
      myname(MO,[M1,M2,M3]),
      validmonth(MO),
      myname(YRS,[Y1,Y2,Y3,Y4]),
      convert(YRS,number,YR),!.

convertdate([D1,SP,M1,M2,M3,SP,Y1,Y2,Y3,Y4],DAY,MO,YR) :-
      myname(DAYS,[D1]),
      convert(DAYS,number,DAY),
      myname(MO,[M1,M2,M3]),
      validmonth(MO),
      myname(YRS,[Y1,Y2,Y3,Y4]),
      convert(YRS,number,YR),!.

convertdate(DATELIST,DAY,MO,YR) :-
      write("RE-ENTER DATE CORRECTLY -- (ex. 01 jan 1989)"), nl,
      get_date(DAY,MO,YR).

get_month(X) :-
      read_record(X),
      validmonth(X).

get_month(X) :-
      read_record(X),
      not validmonth(X).
      write("You MUST enter a 3 letter abbreviation (use lower case only)."),
      nl,write("try again -- "), nl,
      get_month(X).

validmonth(jan).
validmonth(feb).
validmonth(mar).
validmonth(apr).
validmonth(may).
validmonth(jun).
validmonth(jul).
validmonth(aug).
validmonth(sep).
validmonth(oct).
validmonth(nov).
validmonth(dec).

dashes :-
      write("    ------------------------------------------------"),nl.!.
```

```
deploymentheader :-
      write("Enter the deployment database for each squadron based upon"),nl,
      write("the ORE period that begins and/or ends during the schedule"),
       write(" year."),nl,
      dashes.

get_answer(ANS) :-
      read_record(ANS),
       valid_answer(ANS),!.

get_answer(ANS) :-
      write("That was not a valid answer to the question..."),
      write("please type either Y or N -- "),nl,
       get_answer(ANS),!.

valid_answer(ANS) :-
      affirm_answer(ANS).

valid_answer(ANS) :-
      neg_answer(ANS).

affirm_answer(ANS) :-
      make_char_list(ANS,[A|L]),
      convert(A,lower_case,A2).
      A2 = y.

neg_answer(ANS) :-
      make_char_list(ANS,[A|L]),
      convert(A,lower_case,A2),
      A2 = n.

/* The following was taken from LCDR Hutson's  database module.                */
/* The yearend number was increased to account for the extra months at the end of  */
/* the year that the rescheduling may add in stage II (the month_scheduler ).      */

year_to_schedule:-
      yearbegindate(X),
      yearenddate(Y),
      datetodaynumber(X,X2),
      datetodaynumber(Y,Y2),
      del_all_statements(yearbegin/1),
      del_all_statements(yearend/1),
      add_statement(yearbegin(X2),bottom),
      Y3 is Y2 + 90,
      add_statement(yearend(Y3),bottom),!.

squadronlist([vp9,vp19,vp40,vp46,vp47,vp48,vp50]).

eventnames([tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec]).

teamevents([ewp,ewn,en,ewc,ewm,ec]).
```

```
eventcode(ewp,pre_ntpi).
eventcode(ewn,ntpi).
eventcode(ewc,pre_mrci).
eventcode(ewm,mrci).
eventcode(en,natops).
eventcode(ec,command).
eventcode(tr1a,ready_alert).
eventcode(tr1b,ready_alert).
eventcode(tr1c,ready_alert).

sqd_to_number(vp9,9).
sqd_to_number(vp19,19).
sqd_to_number(vp40,40).
sqd_to_number(vp46,46).
sqd_to_number(vp47,47).
sqd_to_number(vp48,48).
sqd _to_number(vp50,50).

endmod  /* module vpinterface */ .
```

VPREADYSEARCH MODULE

This module determines the ready-alert schedule if it is not contained in the file DATABASE.pro and the user wants the computer to compute the ready-alert schedule.

```
/*  The original code for this module was written by Dave Hutson in C-Prolog. It     */
/*   has been converted to M-Prolog.  Capability to schedule 3 readies per squadron  */
/*   has been added.                                                                 */

module vpreadysearch.

export (search1 / 1 , readyevent/4, build_ready_months / 1,build_readyevents /1,
     ready_month/2, ready_half_month / 2 ).

import (member /2, squadronlist /1, xbagof /3, union /3, max /2, processtime /2,
     give_count /2, successor_counter /0, daynumber_to_date /2, month_to_number /3,
     yearend /1, prettyprint /1,  priorevent /4, trialperiod /3, prerequisite /4).

global (status,trialperiod,prerequisite,priorevent,event,search1,
      jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,
      monday,tuesday,wednesday,thursday,friday,saturday,sunday,weekend,
       tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
      vp9,vp19,vp40,vp46,vp47,vp48,vp50).

/*$eject*/
body.

dynamic(totalreadies/1).
dynamic(readyevent/4).
dynamic(ready_month/2).
dynamic(ready_half_month/2).

/* search1 is the top level predicate to make ready assignments.                    */

search1(ReadyL):-
     nl,write("Scheduling the Ready Alerts...."),nl,
     depthsearch([],ReadyL),!,
     write("SEARCH1 RESULTS:"),nl,
     processtime("generating the Ready Alerts",Time),
     give_count(search1,Time),
     write("(Ready Alert Roster):"),nl,
     prettyprint(ReadyL),
     build_readyevents(ReadyL),
     build_ready_months(ReadyL), !.

depthsearch(Start,Ans) :-
     depthsearch2(Start,[Start],[Ans|Statelist]).

depthsearch2(State,Statelist,Statelist) :-
     goalreached1(State), !.
```

```
depthsearch2(State,Statelist,Ans) :-
      successor1(State,Newstate),
      not(member(Newstate,Statelist)),
      successor_counter,
      depthsearch2(Newstate,[Newstate|Statelist],Ans).

/* successor1 predicates are used in depthsearch to assign readies to squadrons.      */
/* One of the next three successors assigns the first ready of the year .             */

successor1(L,[event(Sq,tr1a,S,F)|L]):-
      last_ready_of_previous_year(Sq2,S),
      readyF(S,F),
      pick_nextsquadron(Sq2,Sq),
      preferred_assignment(Sq,S,F).

successor1(L,[event(Sq,tr1a,S,F)|L]):-
      last_ready_of_previous_year(Sq2,S),
      readyF(S,F),
      pick_nextsquadron(Sq2,Sq),
      alternate_assignment1(Sq,S,F).

successor1(L,[event(Sq,tr1a,S,F)|L]):-
      last_ready_of_previous_year(Sq2,S),
      readyF(S,F),
      pick_nextsquadron(Sq2,Sq),
      alternate_assignment2(Sq,S,F).

successor1(L,[event(Sq,tr1a,S,F)|L]):-
      readySF(L,Sq2,S,F),
      pick_nextsquadron(Sq2,Sq),
      not(member(event(Sq,tr1a,_,_),L)),
      preferred_assignment(Sq,S,F).

successor1(L,[event(Sq,tr1a,S,F)|L]):-
      readySF(L,Sq2,S,F),
      pick_nextsquadron(Sq2,Sq),
      not(member(event(Sq,tr1a,_,_),L)),
      alternate_assignment3(Sq,S,F).

successor1(L,[event(Sq,tr1a,S,F)|L]):-
      readySF(L,Sq2,S,F),
      pick_nextsquadron(Sq2,Sq),
      not(member(event(Sq,tr1a,_,_),L)),
      alternate_assignment1(Sq,S,F).

successor1(L,[event(Sq,tr1a,S,F)|L]):-
      readySF(L,Sq2,S,F),
      pick_nextsquadron(Sq2,Sq),
      not(member(event(Sq,tr1a,_,_),L)),
      alternate_assignment4(Sq,S,F).
```

```
successor1(L,[event(Sq,tr1a,S,F)|L]):-
        readySF(L,Sq2,S,F),
        pick_nextsquadron(Sq2,Sq),
        not(member(event(Sq,tr1a,_,_),L)),
        alternate_assignment2(Sq,S,F).

successor1(L,[event(Sq,tr1b,S,F)|L]):-
        readySF(L,Sq2,S,F),
        pick_nextsquadron(Sq2,Sq),
        member(event(Sq,tr1a,S2,F2),L),
        not(member(event(Sq,tr1b,_,_),L)),
        S > F2 + 28,
        preferred_assignment(Sq,S,F).

successor1(L,[event(Sq,tr1b,S,F)|L]):-
        readySF(L,Sq2,S,F),
        pick_nextsquadron(Sq2,Sq),
        member(event(Sq,tr1a,S2,F2),L),
        not(member(event(Sq,tr1b,_,_),L)),
        S > F2 + 28,
        alternate_assignment2(Sq,S,F).

successor1(L,[event(Sq,tr1c,S,F)|L]):-
        readySF(L,Sq2,S,F),
        pick_nextsquadron(Sq2,Sq),
        member(event(Sq,tr1b,S2,F2),L),
        not(member(event(Sq,tr1c,_,_),L)),
        S > F2 + 28,
        preferred_assignment(Sq,S,F).

successor1(L,[event(Sq,tr1c,S,F)|L]):-
        readySF(L,Sq2,S,F),
        pick_nextsquadron(Sq2,Sq),
        member(event(Sq,tr1b,S2,F2),L),
        not(member(event(Sq,tr1c,_,_),L)),
        S > F2 + 28,
        alternate_assignment2(Sq,S,F).

successor1([event(Sq2,E,S1,F1)|L],[event(Sq,tr1a,S,F),event(Sq2,E,S1,F2)|L]):-
        not (Sq = Sq2),
        split_start(S1,F1,S,F2),
        readyF(S,F),
        pick_nextsquadron(Sq2,Sq),
        not(member(event(Sq,tr1a,_,_),L)),
        preferred_assignment(Sq,S,F).
```

```
successor1([event(Sq2,E,S1,F1)|L],[event(Sq,tr1a,S,F),event(Sq2,E,S1,F2)|L]):-
      not (Sq = Sq2),
       split_start(S1,F1,S,F2),
       readyF(S,F),
       pick_nextsquadron(Sq2,Sq),
       not(member(event(Sq,tr1a,_,_),L)),
       alternate_assignment3(Sq,S,F).

successor1([event(Sq2,E,S1,F1)|L],[event(Sq,tr1a,S,F),event(Sq2,E,S1,F2)|L]):-
      not (Sq = Sq2),
       split_start(S1,F1,S,F2),
       readyF(S,F),
       pick_nextsquadron(Sq2,Sq),
       not(member(event(Sq,tr1a,_,_),L)),
       alternate_assignment1(Sq,S,F).

successor1([event(Sq2,E,S1,F1)|L],[event(Sq,tr1a,S,F),event(Sq2,E,S1,F2)|L]):-
      not (Sq = Sq2),
       split_start(S1,F1,S,F2),
       readyF(S,F),
       pick_nextsquadron(Sq2,Sq),
       not(member(event(Sq,tr1a,_,_),L)),
       alternate_assignment4(Sq,S,F).

successor1([event(Sq2,E,S1,F1)|L],[event(Sq,tr1a,S,F),event(Sq2,E,S1,F2)|L]):-
      not (Sq = Sq2),
       split_start(S1,F1,S,F2),
       readyF(S,F),
       pick_nextsquadron(Sq2,Sq),
       not(member(event(Sq,tr1a,_,_),L)),
       alternate_assignment2(Sq,S,F).

/* The next two rules assign a squadron's second ready alert if needed.          */
successor1([event(Sq2,E,S1,F1)|L],[event(Sq,tr1b,S,F),event(Sq2,E,S1,F2)|L]):-
      not (Sq = Sq2),
       split_start(S1,F1,S,F2),
       readyF(S,F),
       pick_nextsquadron(Sq2,Sq),
       not(member(event(Sq,tr1b,_,_),L)),
       preferred_assignment(Sq,S,F).

successor1([event(Sq2,E,S1,F1)|L],[event(Sq,tr1b,S,F),event(Sq2,E,S1,F2)|L]):-
      not (Sq = Sq2),
       split_start(S1,F1,S,F2),
       readyF(S,F),
       pick_nextsquadron(Sq2,Sq),
       not(member(event(Sq,tr1b,_,_),L)),
       alternate_assignment2(Sq,S,F).
```

```
pick_nextsquadron(LastReadySquadron,NextReadySquadron):-
      squadronlist(SL),
      member(NextReadySquadron,SL),
      not(LastReadySquadron = NextReadySquadron).

/* Used by first three successor1 rules.                                    */
last_ready_of_previous_year(LastReadySquadron,NewStart):-
      xbagof(RF1,readyfinishes1(tr1a,RF1),List1),
      xbagof(RF2,readyfinishes1(tr1b,RF2),List2),
      union(List1,List2,List3),
      max(List3,MF),
      (priorevent(LastReadySquadron,tr1a,_,MF);
      priorevent(LastReadySquadron,tr1b,_,MF)),
      NewStart is MF + 1, !.

readyfinishes1(ReadyName,ReadyFinish):-
      priorevent(Sq,ReadyName,ReadyStart,ReadyFinish).

readySF(ListofReadies,LastReadySquadron,NewStart,NewFinish):-
      last_ready(ListofReadies,LastReadySquadron,NewStart),
      readyF(NewStart,NewFinish).

last_ready([event(LastReadySquadron,ReadyName,S,F)|ListofReadies],
        LastReadySquadron,NewStart):-
      NewStart is F + 1,daynumber_to_date(NewStart,X),!.

/* NewStart is first of month.                                              */
readyF(NewStart,NewFinish):-
      trialperiod(tr1,NewStart,NewFinish),!.

/* NewStart is middle of month.                                             */
readyF(NewStart,NewFinish):-
      trialperiod(tr1,S,NewFinish),
      S > NewStart,
      NewFinish < NewStart + 61,!.

/* Ready is no earlier than the third full month after deployment and       */
/* ready is not during ORE vurnerability period.                            */
preferred_assignment(Sq,S,F):-
      prerequisite(Sq,ds0,S2,F2),
      S >= F2 + 58,
      prerequisite(Sq,dv1,S3,F3),
      F < S3,!.
```

```
/* Ready is at least one month after post deployment safety standdown, but before */
/* ORE vurnerability period. */

alternate_assignment1(Sq,S,F):-
      prerequisite(Sq,ds0,S2,F2),
      S >= F2 + 28,
      prerequisite(Sq,dv1,S3,F3),
      F < S3,!.

/* Ready is no earlier than third full month after deployment and */
/* ready can be during first month of ORE vurnerability period. */

alternate_assignment2(Sq,S,F):-
      prerequisite(Sq,ds0,S2,F2),
      S >= F2 + 58,
      prerequisite(Sq,dv1,S3,F3),
      F < F3 - 58,!.

/* Ready is no earlier than third full month after deployment. */

alternate_assignment3(Sq,S,F):-
      prerequisite(Sq,ds1,S2,F2),
      S >= F2 + 58.

/* Ready is at least one month after deployment. */

alternate_assignment4(Sq,S,F):-
      prerequisite(Sq,ds1,S2,F2),
      S >= F2 .

/* Divides three months between two ready alerts. */

split_start(OldStart,OldFinish,SplitStart,SplitFinish):-
      X is OldFinish - OldStart,
      X <= 31,
      SplitFinish is OldFinish + 15,
      SplitStart is SplitFinish + 1,!.

build_readyevents([]):-!.

build_readyevents([event(Sq,E,S,F)|L]):-
      add_statement(readyevent(Sq,E,S,F),bottom),
      build_readyevents(L).

build_ready_months([]):-!.

build_ready_months([event(Sq,E,S,F)|L]):-
      daynumber_to_date(S,[DayS,MonthS,YRS]),
      daynumber_to_date(F,[DayF,MonthF,YRF]),
      MonthS = MonthF,
      month_to_number(MonthS,YRS,MonthNumber),
      add_statement(ready_month(Sq,MonthNumber)),
      build_ready_months(L).
```

```
build_ready_months([event(Sq,E,S,F)|L]):-
      daynumber_to_date(S,[DayS,MonthS,YRS]),
      daynumber_to_date(F,[DayF,MonthF,YRF]),
      not(MonthS = MonthF),
      DayS = 1,
      month_to_number(MonthS,YRS,MonthNumberS),
      month_to_number(MonthF,YRF,MonthNumberF),
      add_statement(ready_month(Sq,MonthNumberS)),
      add_statement(ready_month(Sq,MonthNumberF)),
      add_statement(ready_half_month(Sq,MonthNumberF)),
      build_ready_months(L).

build_ready_months([event(Sq,E,S,F)|L]):-
      daynumber_to_date(S,[DayS,MonthS,YRS]),
      daynumber_to_date(F,[DayF,MonthF,YRF]),
      not(MonthS = MonthF),
      month_to_number(MonthS,YRS,MonthNumberS),
      month_to_number(MonthF,YRF,MonthNumberF),
      add_statement(ready_month(Sq,MonthNumberS)),
      add_statement(ready_month(Sq,MonthNumberF)),
      add_statement(ready_half_month(Sq,MonthNumberS)),
      build_ready_months(L).

goalreached1([event(Sq,E,S,F)|L]):-
      (E = tr1a; E = tr1b; E = tr1c),
      yearend(YE),
      F >= YE-90,   /* yearend(X) has 90 days added for the trialperiods */
      list_length([event(Sq,E,S,F)|L],X),
      add_statement(totalreadies(X)) .  /*  to be used in goalreached2   */

endmod  /* module vpreadysearch */.
```

VPMONTHSEARCH MODULE

This module conducts the stage II search for a schedule with the least number of conflicts, using the months that are as close to the first_pick_months as possible. This module uses Prof. Rowe's modified A* program.

```
module vpmonthsearch.

export   (nopathsearch / 2, usedstate /2, agenda /3, agenda_check /2, search2 /1,
     prunable /4, repeatifagenda / 0, special_less_than /2, usedstate_check / 2,
     cleandatabase / 0, measurework / 0).

import   (prettyprint / 1, deleteone / 3, yearbegindate /1, delete /3, member /2,
     singlemember /2,generate_first_pick / 0, display_statistics /0,ready_half_month /2,
     earmark /3, first_pick_month / 4, month_to_number / 3, subset / 2, ready_month / 2).

global (status,trialperiod,prerequisite,event,dummy,agenda,ready_month, usedstate,
       jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,
       monday,tuesday,wednesday,thursday,friday,saturday,sunday,weekend,
        tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
       vp9,vp19,vp40,vp46,vp47,vp48,vp50).

/*$eject*/
body.

dynamic(usedstate/2).
dynamic(beststate/3).  /* used in vpfinalsearch module also */
dynamic(agenda/3).     /* used in vpfinalsearch module also */
dynamic(counter/1).
dynamic(yrbeginmonth/1).
dynamic(minstate/3).
dynamic(min_conflict_reached/0).

/* A STATE is a list of <event(Sq,Evt,Mo,Yr)>'s                                  */

search2(ANS) :-
     generate_first_pick,
     get_start_state([],START_STATE),
     nopathsearch(START_STATE,ANS) .

/* The result of get_start_state is to build a list (a state) containing all the events that are */
/* first-pick events.  ie, there will be conflicts in this state. but it is a starting place.    */

get_start_state(S1,S2):-
     first_pick_month(Sq,Evt,Mo,Yr),
     del_statement(first_pick_month(Sq,Evt,Mo,Yr)),
     month_to_number(Mo,Yr,MN),
     get_start_state([event(Sq,Evt,MN)|S1],S2).
```

```
get_start_state(S1,S1).

nopathsearch(START,STATE1) :-
      cleandatabase,
      yearbegindate([D,M,Y]),
      month_to_number(M,Y,MN),
      add_statement(yrbeginmonth(MN)),
      add_state(START,START),
      repeatifagenda,
      pick_best_state(STATE,SC,SD),
       write("."),
       add_successors(START,STATE,SC,SD),
      agenda(STATE1,C1,D1),
      del_statement(agenda(STATE1,C1,D1)),
       measurework, !.

pick_best_state(STATE,SC,SD) :-
      add_statement(beststate(dummy,dummy,dummy),top),
      agenda(S,C,D),
      beststate(S2,C2,D2),
      special_less_than(D,D2),
      del_statement(beststate(S2,C2,D2)),
      add_statement(beststate(S,C,D),top),
      fail .

/* The next rule is optional.                                          */

pick_best_state(STATE,SC,SD) :-
       countup(agenda(SX,CX,DX),AC),
      write("Item count BEFORE agenda pruned: "),
      write(AC),nl,fail.

pick_best_state(STATE,SC,SD) :-
      beststate(S,C,D),
      agenda(S2,C2,D2),
      not(S = S2),
      prunable(S2,D2,S,D),
      del_statement(agenda(S2,C2,D2)),
      fail .

/* The next rule is optional.                                          */

pick_best_state(STATE,SC,SD) :-
       countup(agenda(SX,CX,DX),AC),
      write("Item count AFTER agenda pruned: "),
      write(AC),nl,fail.
```

```
/* The next rule is optional.                                          */
pick_best_state(STATE,SC,SD) :-
      beststate(S,C,D),
      write("BESTSTATE: COST = "), write(C),
      write(" ,D = "), write(D), nl,
      prettyprint(S),
      fail .

pick_best_state(STATE,C,D) :-
      beststate(STATE1,C1,D1),
      D1 = dummy,
      del_statement(beststate(STATE1,C1,D1)),
      minstate(STATE,C,D),!.

pick_best_state(STATE,C,D) :-
      beststate(STATE,C,D),
      del_statement(beststate(STATE,C,D)),
      not(D = dummy), ! .

/* ST is the start state and has the first-pick months in it .         */
add_successors(ST,STATE,C,D) :-
      goalreached(STATE,C,D), ! .

add_successors(ST,STATE,C,D) :-
      reset_min_conflict,
      successor(ST,STATE,NEWSTATE),
      del_all_statements(min_conflict_reached/0),
      add_state(ST,NEWSTATE),
      fail .

add_successors(ST,STATE,C,D) :-
      agenda(STATE,C,D),
      del_statement(agenda(STATE,C,D)),
      add_statement(usedstate(STATE,C),top),
      fail .

add_successors(ST,STATE,C,D) :-
      min_conflict_reached,
      del_statement(min_conflict_reached),
      add_statement(minstate(STATE,C,D),top),
      fail .

reset_min_conflict :-
      add_statement(min_conflict_reached),!.
```

```
add_state(ST,NEWSTATE) :-
      compute_cost(ST,NEWSTATE,CNEW), !,
      agenda_check(NEWSTATE,CNEW), !,
      usedstate_check(NEWSTATE,CNEW), !,
      eval(NEWSTATE,ENEW),
      D is ENEW + CNEW,
      add_statement(agenda(NEWSTATE,CNEW,D), top), ! .

add_state(ST,NEWSTATE) :-
      not(compute_cost(CNEW,NEWSTATE)),
      write("Warning: your cost function failed on path list "),
      write(PATHLIST), nl, ! .

add_state(ST,NEWSTATE) :-
      not eval(NEWSTATE,ENEW),
      write("Warning: your evaluation function failed on state "),
      write(NEWSTATE), nl, ! .

agenda_check(S,C) :-
      agenda(S,C2,D2),
      del_statement(agenda(S,C2,D2)), ! .

agenda_check(S,C) :-
      agenda(S,C2,D2), !, fail .

agenda_check(S,C) .

usedstate_check(S,C) :-
      usedstate(S,C2),
      del_statement(usedstate(S,C2)),
      add_statement(usedstate(S,C),top), !.

usedstate_check(S,C) :-
      usedstate(S,C2), !,

usedstate_check(S,C) .

repeatifagenda .

repeatifagenda :-
      agenda(X,Y,Z), repeatifagenda .

special_less_than(X,dummy) :- ! .

special_less_than(X,Y) :- X<Y .

cleandatabase :-
      del_all_statements(agenda/3),
      del_all_statements(usedstate/2),
      del_all_statements(beststate/3),
      del_all_statements(counter/1).
```

```
measurework :-
      countup(agenda(X,C,D),NA),
      countup(usedstate(S,C),NB),
      write(NA), write(" incompletely examined state(s) and "),
      write(NB), write(" examined state(s)"),nl, ! .

countup(P,N) :-
      add_statement(counter(0)),
      evaluate(P), /* call(P), */
      counter(K),
      del_statement(counter(K)),
      K2 is K+1,
      add_statement(counter(K2)),
      fail .

countup(P,N) :-
      counter(N),
      del_statement(counter(N)), ! .

eval(STATE,E) :-
      count_conflicts(STATE,E),!.

count_conflicts([],0) .

count_conflicts(STATE,CONFLICTS) .-
      find_conflict(STATE,EVENT),
      delete(EVENT,STATE,STATE2),
      count_conflicts(STATE2,CONFLICTS2),
      CONFLICTS is CONFLICTS2 + 1.

count_conflicts(STATE,0) .

/* When successor does returns it returns a new state.  Find-conflict will find the next   */
/* event in conflict if backtracked to.                                                    */

successor(StartSTATE,STATE,[ReSched_Event|STATE2]):-
      find_conflict(STATE,EVENT),
      deleteone(EVENT,STATE,STATE2),
      get_FP_MN(StartSTATE,EVENT,FirstPickMN),
      reschedule(EVENT,FirstPickMN,STATE,ReSched_Event).

/* The goal is reached when the number of conflicts is zero or as low as it will be.       */

goalreached(STATE,C,D) :-
      Diff is D - C,
      Diff = 0.

goalreached(STATE,C,D) :-
      minstate(STATE,C,D),
      add_statement(agenda(STATE,C,D),top).
```

```
find_conflict(STATE,EVENT) :-
      member(EVENT,STATE),
      deleteone(EVENT,STATE,STATE2),
      conflict(STATE2,EVENT).

conflict(STATE,event(Sq,en,MN)) :-   /* The NATOPS is too long to sched w/ RA.   */
      ready_month(Sq,MN),!.

conflict(STATE,event(Sq,Insp,MN)) :-
      ready_month(Sq,MN),!,
      not(ready_half_month(Sq,MN)),!.

conflict(STATE,event(Sq,en,MN)) :-
      member(event(Sq,AnyInsp,MN),STATE),!.

conflict(STATE,event(Sq,ec,MN)) :-
      member(event(Sq,AnyInsp,MN),STATE),!.

conflict(STATE,event(Sq,ewc,MN)) :-
      member(event(Sq,AnyInsp,MN),STATE),!,
      not(AnyInsp = ewm),!.

conflict(STATE,event(Sq,ewm,MN)) :-
      member(event(Sq,AnyInsp,MN),STATE),
      not(AnyInsp = ewc),!.

conflict(STATE,event(Sq,ewp,MN)) :-
      member(event(Sq,AnyInsp,MN),STATE),
      not(AnyInsp = ewn),!.

conflict(STATE,event(Sq,ewn,MN)) :-
      member(event(Sq,AnyInsp,MN),STATE),
      not(AnyInsp = ewp),!.

conflict(STATE,event(Sq,Insp,MN)) :-
      member(event(AnySq,Insp,MN),STATE),!.

conflict(STATE,event(Sq,ewp,MN)) :-
      member(event(Sq,ewn,MN2),STATE),
      MN > MN2,!.

conflict(STATE,event(Sq,ewp,MN)) :-
      member(event(Sq,ewn,MN2),STATE),
      Diff is MN2 - 1,
      MN < Diff,!.

conflict(STATE,event(Sq,ewn,MN)) :-
      member(event(Sq,ewp,MN2),STATE),
      MN < MN2,!.
```

```
conflict(STATE,event(Sq,ewc,MN)) :-
      member(event(Sq,ewm,MN2),STATE),
      MN > MN2,!.

conflict(STATE,event(Sq,ewc,MN)) :-
      member(event(Sq,ewm,MN2),STATE),
      Diff is MN2 - 1,
      MN < Diff,!.

conflict(STATE,event(Sq,ewm,MN)) :-
      member(event(Sq,ewc,MN2),STATE),
      MN < MN2,!.


get_FP_MN(StartSTATE,event(Sq,Insp,MN),FirstPickMN) :-
      singlemember(event(Sq,Insp,FirstPickMN),StartSTATE),!.

/* Reschedule will not succeed if the event can't be scheduled without  confict during    */
/* window of +/- 3 months of first-pick-month                                             */
reschedule(event(Sq,Insp,MN),EventFirstPickMN,STATE,event(Sq,Insp,PriorMN)):-
      EventFirstPickMN = MN,
      PriorMN is EventFirstPickMN - 1,
      yrbeginmonth(YrBeginMonth),
      PriorMN >= YrBeginMonth,
       prerequisite(Sq,ds0,S,F),
       daynumber_to_date(S,[Sday,Smo,Syr]),
       month_to_number(Smo,Syr,SSMN),
       PriorMN > SSMN,                          /* Added  to prevent scheduling during ssd. */
       not(find_conflict(STATE,event(Sq,Insp,PriorMN))),!.

reschedule(event(Sq,Insp,MN),EventFirstPickMN,STATE,event(Sq,Insp,PostMN)):-
      MN > EventFirstPickMN - 2,
      MN < EventFirstPickMN + 1,
      PostMN is EventFirstPickMN  + 1,
      earmark(Sq,Insp,[Day,Month,Year]),
      month_to_number(Month,Year,DDM),   /* Drop-Dead Month */
      PostMN = DDM,
      not(find_conflict(STATE,event(Sq,Insp,PostMN))),!.

reschedule(event(Sq,Insp,MN),EventFirstPickMN,STATE,event(Sq,Insp,PostMN)):-
      MN > EventFirstPickMN - 2,
      MN < EventFirstPickMN + 1,
      PostMN is EventFirstPickMN  + 1,
      not earmark(Sq,Insp,_).   /* case of no earmark input */
      not(find_conflict(STATE,event(Sq,Insp,PostMN))),!.
```

```
reschedule(event(Sq,Insp,MN),EventFirstPickMN,STATE,event(Sq,Insp,PriorMN)):-
      MN > EventFirstPickMN - 2 ,
      MN < EventFirstPickMN + 2,
      PriorMN is EventFirstPickMN - 2,
      yrbeginmonth(YrBeginMonth),
      PriorMN >= YrBeginMonth,
       prerequisite(Sq,ds0,S,F),
       daynumber_to_date(S,[Sday,Smo,Syr]),
       month_to_number(Smo,Syr,SSMN),
       PriorMN > SSMN,
       not(find_conflict(STATE,event(Sq,Insp,PriorMN))),!.

reschedule(event(Sq,Insp,MN),EventFirstPickMN,STATE,event(Sq,Insp,PostMN)):-
      MN > EventFirstPickMN - 3 ,
      MN < EventFirstPickMN + 2,
      PostMN is EventFirstPickMN + 2,
      earmark(Sq,Insp,[Day,Month,Year]),
      month_to_number(Month,Year,DDM),                    /* Drop-Dead Month .  */
      PostMN <= DDM,
      not(find_conflict(STATE,event(Sq,Insp,PostMN))),!.

reschedule(event(Sq,Insp,MN),EventFirstPickMN,STATE,event(Sq,Insp,PostMN)):-
      MN > EventFirstPickMN - 3 ,
      MN < EventFirstPickMN + 2,
      PostMN is EventFirstPickMN + 2,
      not earmark(Sq,Insp,_),                             /* case of no earmark input */
      not(find_conflict(STATE,event(Sq,Insp,PostMN))),!.

reschedule(event(Sq,Insp,MN),EventFirstPickMN,STATE,event(Sq,Insp,PriorMN)):-
      MN > EventFirstPickMN - 3 ,
      MN < EventFirstPickMN + 3,
      PriorMN is EventFirstPickMN - 3,
      PriorMN =/= MN,
      yrbeginmonth(YrBeginMonth),
      PriorMN >= YrBeginMonth,
       prerequisite(Sq,ds0,S,F),
       daynumber_to_date(S,[Sday,Smo,Syr]),
       month_to_number(Smo,Syr,SSMN),
       PriorMN > SSMN,
       not(find_conflict(STATE,event(Sq,Insp,PriorMN))),!.

reschedule(event(Sq,Insp,MN),EventFirstPickMN,STATE,event(Sq,Insp,PostMN)):-
      PostMN is EventFirstPickMN + 3,
      PostMN =/= MN,
      earmark(Sq,Insp,[Day,Month,Year]),
      month_to_number(Month,Year,DDM),                    /* Drop-Dead Month */
      PostMN <= DDM,
      not(find_conflict(STATE,event(Sq,Insp,PostMN))).
```

```
reschedule(event(Sq,Insp,MN),EventFirstPickMN,STATE,event(Sq,Insp,PostMN)):-
      PostMN is EventFirstPickMN + 3,
      PostMN =/= MN,
      not earmark(Sq,Insp,_), /* case of no earmark input */
      not(find_conflict(STATE,event(Sq,Insp,PostMN))),!.

/* If ewc or ewp are in conflict then put them in month with real inspection if it is not in a */
/* conflict .                                                                                   */

reschedule(event(Sq,ewc,MN),_,STATE,event(Sq,ewc,MN1)):-
      singlemember(event(Sq,ewm,MN1),STATE),
      MN =/= MN1,
      not find_conflict(STATE,event(Sq,ewm,MN1)), !.

reschedule(event(Sq,ewp,MN),_,STATE,event(Sq,ewp,MN1)):-
      singlemember(event(Sq,ewn,MN1),STATE),
      MN =/= MN1,
      not find_conflict(STATE,event(Sq,ewn,MN1)), !.

compute_cost(STARTSTATE,[],0):- !.
compute_cost(STARTSTATE,[event(Sq,Insp,MN)|STATE],Cost) :-
      singlemember(event(Sq,Insp,BestMonth),STARTSTATE),
      Diff is MN - BestMonth,
      cost_function(Diff,Cost1),
      compute_cost(STARTSTATE,STATE,Cost2),
      Cost is Cost1 + Cost2.

cost_function(0,0).
cost_function(1,1).
cost_function(2,2).
cost_function(3,3).
cost_function(-1,0.5e0).
cost_function(-2,1.5e0).
cost_function(-3,2.5e0).

/* Prunable determines degree of optimization by purging agenda of selected items.      */

prunable(STATE,D,BESTSTATE,Dbest) :-
      k_factor(K),
      D > Dbest + K, !.

prunable(STATE,D,BESTSTATE,Dbest) :-
      check_permutation(STATE,BESTSTATE),!.

check_permutation(STATE1,STATE2) :-
      subset(STATE1,STATE2),
      subset(STATE2,STATE1),!.

k_factor(0).

endmod /* vpmonthsearch */ .
```

VPFIRSTPICK MODULE

This module computes the first_pick_month for each event to be scheduled.

```
module vpfirstpick .

export( generate_first_pick / 0, first_pick_month / 4, month_to_number / 3,
     number_to_month / 3 ).

import( eventnames / 1, squadronlist / 1, prerequisite / 4,datetodaynumber/2,
     daynumber_to_date /2, earmark /3, member /2, priorevent / 4, yearbegindate/ 1,
     yearenddate / 1, yearend / 1).

global (status,trialperiod,prerequisite,priorevent,
      jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,
      monday,tuesday,wednesday,thursday,friday,saturday,sunday,weekend,
     month_to_number / 3, number_to_month / 3 ).
     priorevent / 4, yearbegindate/ 1, yearenddate / 1, yearend / 1).
       tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
      vp9,vp19,vp40,vp46,vp47,vp48,vp50).

/*$eject*/
body.

dynamic(first_pick_month/4).
dynamic(tot_evt_count/1).

generate_first_pick :-
      write("Finding the 'first-pick' month for each inspection ..."),nl,nl,
     eventnames(InspectionList),
     member(Inspection,InspectionList),
     squadronlist(SqList),
     member(Sq,SqList),
     evt_ok_to_sched(Sq,Inspection),
     generate_first_pick_month(Sq,Inspection),
     fail.

generate_first_pick :-
      write("The 'first-pick' months follow:"),nl,
      first_pick_month(Sq,Insp,MO,YR),
      write(Sq),write_tab(7),
      write(Insp),write_tab(12),
      write(MO),write_tab(17),
      write(YR),nl,
     fail.

generate_first_pick .
```

```
/* Check if event has already been conducted during the current training cycle.   */

evt_ok_to_sched(Sq,Insp) :-
      evt_not_ready(Sq,Insp),
      not(priorevent(Sq,Insp,Start,Finish)),!.

evt_ok_to_sched(Sq,ewp) :-
      priorevent(Sq,ewn,StartDate,PEDN),
      earmark(Sq,ewn,EMDate),
      datetodaynumber(EMDate,EMDN),
      prerequisite(Sq,dr1,DrSDN,DrFDN),
      yearend(YE),
      PEDN <= DrSDN,
      EMDN <= YE,
      EMDN >= DrFDN, !.

evt_ok_to_sched(Sq,Insp) :-
      evt_not_ready(Sq,Insp),
      priorevent(Sq,Insp,StartDate,PEDN),
      earmark(Sq,Insp,EMDate),
      datetodaynumber(EMDate,EMDN),
      prerequisite(Sq,dr1,DrSDN,DrFDN),
      yearend(YE),
      PEDN <= DrSDN,
      EMDN <= YE,
      EMDN >= DrFDN, !.

evt_not_ready(Sq,Insp) :-
      not(Insp = tr1a),
      not(Insp = tr1b),
      not(Insp = tr1c),!.

generate_first_pick_month(Sq,ewn) :-
      earmark(Sq,ewn,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      MN2 is MN-0,
      year_begin_month_num(YB),
      MN2 >= YB,
      year_end_month_num(YE),
      MN2 <= YE,
      number_to_month(Month2,Year2,MN2),
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,ewn,Month2,Year2)),!.
```

```
generate_first_pick_month(Sq,ewp) :-
      earmark(Sq,ewn,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      MN2 is MN-1,
      year_begin_month_num(YB),
      MN2 >= YB,
      year_end_month_num(YE),
      MN2 <= YE,
      number_to_month(Month2,Year2,MN2),
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,ewp,Month2,Year2)),!.

generate_first_pick_month(Sq,ewc) :-
      prerequisite(Sq,dr1,Startdays,Finishdays),
      daynumber_to_date(Startdays,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      MN2 is MN-4,
      earmark(Sq,ewc,[Day3,Month3,Year3]),
      month_to_number(Month3,Year3,MN3),
      MN2 <= MN3,
      year_begin_month_num(YB),
      MN2 >= YB,
      year_end_month_num(YE),
      MN2 <= YE,
      number_to_month(Month2,Year2,MN2),
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,ewc,Month2,Year2)),!.

generate_first_pick_month(Sq,ewc) :-
      earmark(Sq,ewc,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      year_begin_month_num(YB),
      MN >= YB,
      year_end_month_num(YE),
      MN <= YE,
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,ewc,Month,Year)),!.
```

```
generate_first_pick_month(Sq,ewm) :-
      prerequisite(Sq,dr1,Startdays,Finishdays),
      daynumber_to_date(Startdays,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      MN2 is MN-3,
      earmark(Sq,ewm,[Day3,Month3,Year3]),
      month_to_number(Month3,Year3,MN3),
      MN2 <= MN3,
      year_begin_month_num(YB),
      MN2 >= YB,
      year_end_month_num(YE),
      MN2 <= YE,
      number_to_month(Month2,Year2,MN2),
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,ewm,Month2,Year2)),!.

generate_first_pick_month(Sq,ewm) :-
      earmark(Sq,ewm,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      year_begin_month_num(YB),
      MN >= YB,
      year_end_month_num(YE),
      MN <= YE,
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,ewm,Month,Year)),!.

generate_first_pick_month(Sq,ec) :-
      prerequisite(Sq,dr1,Startdays,Finishdays),
      daynumber_to_date(Startdays,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      MN2 is MN-2,
      earmark(Sq,ec,[Day3,Month3,Year3]),
      month_to_number(Month3,Year3,MN3),
      MN2 <= MN3,
      year_begin_month_num(YB),
      MN2 >= YB,
      year_end_month_num(YE),
      MN2 <= YE,
      number_to_month(Month2,Year2,MN2),
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,ec,Month2,Year2)),!.

generate_first_pick_month(Sq,ec) :-
      earmark(Sq,ec,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      year_begin_month_num(YB),
      MN >= YB,
      year_end_month_num(YE),
      MN <= YE,
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,ec,Month,Year)),!.
```

```
/* Next rule in case no earmark is input for CI.                    */

generate_first_pick_month(Sq,ec) :-
      prerequisite(Sq,dr1,Startdays,Finishdays),
      daynumber_to_date(Startdays,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      MN2 is MN-2,
      year_begin_month_num(YB),
      MN2 >= YB,
      year_end_month_num(YE),
      MN2 <= YE,
      number_to_month(Month2,Year2,MN2),
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,ec,Month2,Year2)),!.

generate_first_pick_month(Sq,en) :-
      prerequisite(Sq,dr1,Startdays,Finishdays),
      daynumber_to_date(Startdays,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      MN2 is MN-7,
      earmark(Sq,en,[Day3,Month3,Year3]),
      month_to_number(Month3,Year3,MN3),
      MN2 <= MN3,
      year_begin_month_num(YB),
      MN2 >= YB,
      year_end_month_num(YE),
      MN2 <= YE,
      number_to_month(Month2,Year2,MN2),
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,en,Month2,Year2)),!.

generate_first_pick_month(Sq,en) :-
      earmark(Sq,en,[Day,Month,Year]),
      month_to_number(Month,Year,MN),
      year_begin_month_num(YB),
      MN >= YB,
      year_end_month_num(YE),
      MN <= YE,
      inc_tot_evt_count,
      add_statement(first_pick_month(Sq,en,Month,Year)),!.

month_to_number(Month,Year,MonthNumber) :-
      month_num(Month,MN),
      MonthNumber is MN + (Year*12).

number_to_month(Month,Year,MonthNumber) :-
      Year is MonthNumber div 12,
      MN is MonthNumber mod 12,
      month_num(Month,MN).
```

```
month_num(jan,0).
month_num(feb,1).
month_num(mar,2).
month_num(apr,3).
month_num(may,4).
month_num(jun,5).
month_num(jul,6).
month_num(aug,7).
month_num(sep,8).
month_num(oct,9).
month_num(nov,10).
month_num(dec,11).

year_begin_month_num(MN) :-
      yearbegindate([Day,Month,YR]),
      month_to_number(Month,YR,MN).

year_end_month_num(MN) :-
      yearenddate([Day,Month,YR]),
      month_to_number(Month,YR,MN).

inc_tot_evt_count :-
      tot_evt_count(X),
      Y is X+1,
      del_statement(tot_evt_count(X)),
      add_statement(tot_evt_count(Y)),!.

inc_tot_evt_count :-
      add_statement(tot_evt_count(1)),!.

endmod  /* module vpfirstpick */ .
```

VPFINALSEARCH MODULE

This module does the stage III search for the optimum final schedule.

```
module vpfinalsearch.

export (generate_final_schedule /3 ).

import (max / 2, yearbegin / 1, datetodaynumber / 2, singlemember / 2,
     number_to_month / 3, daysofmonth / 2,difference_between_dates2 / 3, agenda / 3,
     readyevent/4,prettyprint /1, database_conversion /0, cleandatabase / 0,
     repeatifagenda / 0,squadronlist / 1, measurework / 0, agenda_check / 2,
     usedstate_check / 2, successor_counter / 0, sortmonthevent / 2,
     special_less_than /2,prunable /4, prerequisite /4,  trialperiod /3, member /2,
     sortevent /2,priorevent / 4, open / 2, tell / 2, told / 1, teamevents /1, yearend /1).

 gloabal (jan,feb,mar, apr,may,jun,jul,aug,sep,oct,nov,dec,
     monday,tuesday,wednesday,thursday,friday,saturday,sunday,weekend,
     status,trialperiod,prerequisite,event,dummy,usedstate,agenda, readyevent,
       tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
       vp9,vp19,vp40,vp46,vp47,vp48,vp50).

local(state).

/*$eject*/
body.

dynamic(total_events/1).
dynamic(beststate/2).
dynamic(monevt/3).
dynamic(sccsrfound/0).
dynamic(min_cost_reached/0).
dynamic(oldcost/1).
dynamic(index/1).
dynamic(state/2).
dynamic(cost/2).
dynamic(scost/3).
dynamic(tcost/3).

generate_final_schedule(StartList,MonthList,FinalList) :-
     write("Generating the final schedule ....."),nl,
     list_length(StartList,L1),
     list_length(MonthList,L2),
     Total is L1 + L2,
     add_statement(total_events(Total)),
     sortmonthevent(MonthList,SortedList),
     assert_monevts(SortedList),
     hillclimbsrch(StartList,FinalList).
```

```
assert_monevts([]).
assert_monevts([event(SQ,INSP,MN)|LIST]) :-
      add_statement(monevt(SQ,INSP,MN),bottom),
      assert_monevts(LIST).

hillclimbsrch(START,STATE) :-
      cleandatabase2,
      resetindex,
      index(IndexS),
      initialize_start(IndexS,START),
      add_state2(start,IndexS),
      repeatifagenda,
      pick_best_state2(INDEX),
      write("."),
      del_all_statements(oldcost/1),
      add_statement(oldcost(dummy)),
      add_successors2(INDEX),
      del_all_statements(sccsrfound/0),
      agenda(INDEX,C,D),
      del_statement(agenda(INDEX,C,D)),
      state(INDEX,STATE),
      measurework.


pick_best_state2(INDEX) :-
      agenda(INDEX,C,D) .

add_successors2(INDEX) :-
      goalreached2(INDEX), +.

add_successors2(INDEX) :-
      del_all_statements(sccsrfound/0),
      del_all_statements(min_cost_reached/0),
      state(INDEX,STATE),
      successor2a(STATE,NewSTATE),
      incindex,
      index(NewINDEX),
      add_statement(state(NewINDEX,NewSTATE)),
      check_cost(INDEX,NewINDEX),
      add_state2(INDEX,NewINDEX),
      successor_counter,

add_successors2(INDEX) :-
      del_statement(agenda(INDEX,C,D)),
      add_statement(usedstate(INDEX,C)),

incindex:-
      index(OLDI),
      del_statement(index(OLDI)),
      NewI is OLDI + 1,
      add_statement(index(NewI)),+.
```

```
resetindex :-
      del_all_statements(index/1),
      add_statement(index(0)),+.

del_dynamics(INDEX) :-
      del_statement(state(INDEX,STATE)).

initialize_start(INDEX,START) :-
      add_statement(state(INDEX,START)),
      add_statement(cost(INDEX,0)),
      init_scost(INDEX),
      init_tcost(INDEX),+.

init_scost(INDEX) :-
      squadronlist(SQLIST),
      member(SQ,SQLIST),
      add_statement(scost(INDEX,SQ,0)),
      fail.

init_scost(INDEX) :- !.

init_tcost(INDEX) :-
      teamevents(EVTLIST),
      member(EVT,EVTLIST),
      add_statement(tcost(INDEX,EVT,0)),
      fail.

init_tcost(INDEX) :- !.

cost2(INDEX,NEWINDEX,NCost) :-
      state(NEWINDEX,[event(SQ,EVT,S,F)|STATE]),
      teamevents(EVTLIST),
      singlemember(EVT,EVTLIST),
      copycost(INDEX,NEWINDEX),
      squadroncost(SQ,[event(SQ,EVT,S,F)|STATE],SCOST),
      scost(NEWINDEX,SQ,X),
      del_statement(scost(NEWINDEX,SQ,X)),
      add_statement(scost(NEWINDEX,SQ,SCOST)),
      teamcost(EVT,[event(SQ,EVT,S,F)|STATE],TCOST),        /* EVT is a teamevt */
      tcost(NEWINDEX,EVT,X1),
      del_statement(tcost(NEWINDEX,EVT,X1)),
      add_statement(tcost(NEWINDEX,EVT,TCOST)),
      computecost(NEWINDEX,NCost), +.
```

```
cost2(INDEX,NEWINDEX,NCost) :-
      state(NEWINDEX,[event(SQ,EVT,S,F)|STATE]),
      copycost(INDEX,NEWINDEX),
      squadroncost(SQ,[event(SQ,EVT,S,F)|STATE],SCOST),
      scost(NEWINDEX,SQ,X),
      del_statement(scost(NEWINDEX,SQ,X)),
      add_statement(scost(NEWINDEX,SQ,SCOST)),!,
      computecost(NEWINDEX,NCost), +.

copycost(INDEX,NEWINDEX) :-
      copyscost(INDEX,NEWINDEX),
      copytcost(INDEX,NEWINDEX),!.

copyscost(INDEX,NEWINDEX) :-
     squadronlist(SQLIST),
     member(SQ,SQLIST),
     scost(INDEX,SQ,SCOST),
     add_statement(scost(NEWINDEX,SQ,SCOST)),
     fail .

copyscost(INDEX,NEWINDEX) :- !.

copytcost(INDEX,NEWINDEX) :-
     teamevents(EVTLIST),
     member(EVT,EVTLIST),
     tcost(INDEX,EVT,TCOST),
     add_statement(tcost(NEWINDEX,EVT,TCOST)),
     fail .

copytcost(INDEX,NEWINDEX) :- !.

computecost(INDEX,Cost) :-
      squadronlist(SQLIST),
      compscost(INDEX,SQLIST,SCOST),
      teamevents(EVTLIST),!,
      comptcost(INDEX,EVTLIST,TCOST),
      Cost is SCOST + TCOST.

compscost(INDEX,[],0) :- !.

compscost(INDEX,[SQ|SQLIST],SCOST) :-
      scost(INDEX,SQ,Cost1),
      compscost(INDEX,SQLIST,Cost2),
      SCOST is Cost1 + Cost2.

comptcost(INDEX,[],0) :- !.

comptcost(INDEX,[EVT|EVTLIST],TCOST) :-
      tcost(INDEX,EVT,Cost1),
      comptcost(INDEX,EVTLIST,Cost2),
      TCOST is Cost1 + Cost2.
```

```
squadroncost(Sq,State,Scost):-
      bag_of(Occurance,groupsquadron(Sq,State,Occurance),List),
      sortevent(List,[X|SortedList]),
      spiece_cost(start,X,Scost2x),
      scost2x([X|SortedList],Scost3),
      Scost is Scost2x + Scost3.

teamcost(Eventname,State,Tcost):-
      teamevents(TE),
      member(Eventname,TE),
      bag_of(Occurance,groupteam(Eventname,State,Occurance),List),
      sortevent(List,SortedList),
      tcost2x(SortedList,Tcost).

check_cost(INDEX,NewINDEX) :-
      getcosts(INDEX,NewINDEX,NewC,OldC),
      special_less_than(NewC,OldC),
      del_statement(oldcost(OldC)),
      add_statement(oldcost(NewC)),
      remove_agenda_item(OldC),

check_cost(INDEX,NewINDEX) :-
      add_statement(min_cost_reached), - .                    /* "-" is same as "!,fail" */

remove_agenda_item(OldC) :-
      OldC = dummy.

remove_agenda_item(OldC) :-
      agenda(I,OldC,D),                                       /* Remove the higher cost. */
      del_statement(agenda(I,OldC,D)).

getcosts(INDEX,NewINDEX,NewC,OldC):-
      oldcost(OldC),
      cost2(INDEX,NewINDEX,NewC),+.

add_state2(start,NewINDEX) :-
      state(NewINDEX,STATE),
      eval2(STATE,Enew),
      D is Enew,
      add_statement(agenda(NewINDEX,0,D)), +.

add_state2(INDEX,NewINDEX) :-
      oldcost(Cnew),!,
      agenda_check(NewINDEX,Cnew), !,
      usedstate_check(NewINDEX,Cnew), !,
      state(NewINDEX,NewSTATE),
      eval2(NewSTATE,Enew),
      D is Enew + Cnew,
      add_statement(agenda(NewINDEX,Cnew,D)), +.
```

```
add_state2(INDEX,NewINDEX) :-
      state(NewINDEX,NewSTATE),
      not eval2(NewSTATE,Enew),
      write("Warning: your evaluation function failed on state "),
      write(NewINDEX), nl, +.

goalreached2(INDEX) :-
      state(INDEX,STATE),
      total_events(Total),
      list_length(STATE,Length),!,
      Total = Length .

/* For successor2a to pick the events chronologically monevt's must be asserted into    */
/* the data base correctly.                                                              */

successor2a(STATE,NewSTATE) :-
      monevt(SQ,ec,MN),
      not member(event(SQ,ec,S,F),STATE),
      successor2(SQ,ec,MN,STATE,NewSTATE),+.

successor2a(STATE,NewSTATE) :-
      monevt(SQ,EVT,MN),
      not member(event(SQ,EVT,S,F),STATE),!,
      write("adding "),write(SQ),write(" "),write(EVT),nl,
      successor2(SQ,EVT,MN,STATE,NewSTATE).

successor2(SQ,ec,MN,STATE,[event(SQ,ec,Start,Finish)|STATE]) :-
      prerequisite(SQ,dr1,DS,DF),
      IdealStart is DS - 45,
      number_to_month(MO,YR,MN),
      datetodaynumber([1,MO,YR],MOStart),
      bag_of(S,validstart(IdealStart,MOStart,S),SList),
      max(SList,Start),
      trialperiod(ec,Start,Finish),+.          /* Cut used because there's only one choice. */

successor2(SQ,ewp,MN,STATE,[event(SQ,ewp,Start,Finish)|STATE]) :-
      monevt(SQ,ewn,MN),                       /* Find ewp and ewn in same month. */
      not sccsrfound,
      get_moinfo(MN,MOStart,MOEnd),
      trialperiod(ewp,Start,Finish),
      not min_cost_reached,
      Finish >= MOStart,
      Start < MOStart + 7,
      not readyconflict(SQ,Start,Finish),
      not realeventconflict(ewp,Start,Finish,STATE),
      add_statement(sccsrfound).
```

```
successor2(SQ,ewn,MN,STATE,[event(SQ,ewn,Start,Finish)|STATE]) :-
      monevt(SQ,ewp,MN),                    /* Find ewp and ewn in same month. */
      not sccsrfound,
      get_moinfo(MN,MOStart,MOEnd),
      trialperiod(ewn,Start,Finish),
      not min_cost_reached,
      Start > MOStart + 11,
      Start <= MOEnd,
      not readyconflict(SQ,Start,Finish),
      not realeventconflict(ewn,Start,Finish,STATE),
      add_statement(sccsrfound).

successor2(SQ,ewc,MN,STATE,[event(SQ,ewc,Start,Finish)|STATE]) :-
      monevt(SQ,ewm,MN),                    /* Find ewc and ewm in same month. */
      not sccsrfound,
      get_moinfo(MN,MOStart,MOEnd),
      trialperiod(ewc,Start,Finish),
      not min_cost_reached,
      Finish >= MOStart,
      Start < MOStart + 7,
      not readyconflict(SQ,Start,Finish),
      add_statement(sccsrfound).

successor2(SQ,ewm,MN,STATE,[event(SQ,ewm,Start,Finish)|STATE]) :-
      monevt(SQ,ewc,MN),                    /* Find ewc and ewm in same month. */
      not sccsrfound,
      get_moinfo(MN,MOStart,MOEnd),
      trialperiod(ewm,Start,Finish),
      not min_cost_reached,
      Start > MOStart + 11,
      Start <= MOEnd,
      not readyconflict(SQ,Start,Finish),
      add_statement(sccsrfound).

successor2(SQ,INSP,MN,STATE,[event(SQ,INSP,Start,Finish)|STATE]) :-
      not sccsrfound,
      get_moinfo(MN,MOStart,MOEnd),
      trialperiod(INSP,Start,Finish),
      not min_cost_reached,
      Finish >= MOStart,
      Start <= MOEnd ,
      not readyconflict(SQ,Start,Finish),
      not otherevtconflict(SQ,Start,Finish,STATE),
      not realeventconflict(INSP,Start,Finish,STATE),
      add_statement(sccsrfound).
```

```
/* The next rule is the 'catch-all'.                                    */
successor2(SQ,INSP,MN,STATE,[event(SQ,INSP,Start,Finish)|STATE]) :-
      get_moinfo2(MN,MOStart,MOEnd),
      trialperiod(INSP,Start,Finish),
      not min_cost_reached,
      Finish >= MOStart,
      Start <= MOEnd .

get_moinfo(MN,MOStart,MOEnd):-
      number_to_month(MO,YR,MN),
      datetodaynumber([1,MO,YR],MOStart),
      daysofmonth(MO,LastDay),
      datetodaynumber([LastDay,MO,YR],MOEnd),+.

get_moinfo2(MN,MOStart,MOEnd):-
      not sccsrfound./* prevents using rule if 1st rule worked */
      number_to_month(MO,YR,MN),
      datetodaynumber([1,MO,YR],MOStart)
      daysofmonth(MO,LastDay),
      datetodaynumber([LastDay,MO,YR],MOEnd),+.

validstart(Ideal,MOStart,Start) :-
      trialperiod(ec,Start,F),
      Start <= Ideal,
      Start >= MOStart.

readyconflict(SQ,S,F) :-
      readyevent(SQ,E,RS,RF),
      RS <= S,
      RF >= S.

readyconflict(SQ,S,F) :-
      readyevent(SQ,E,RS,RF),
      RS <= F,
      RF >= F.

otherevtconflict(SQ,S,F,STATE) :-
      member(event(SQ,E,S1,F1),STATE),
      S1 <= S,
      F1 >= S.

otherevtconflict(SQ,S,F,STATE) :-
      member(event(SQ,E,S1,F1),STATE),
      S1 <= F,
      F1 >= F.

otherevtconflict(SQ,S,F,STATE) :-
      member(event(SQ,E,S1,F1),STATE),
      S1 >= S,
      S1 <= F.
```

```
realeventconflict(ewp,Start,Finish,STATE):-
      member(event(SQ,ewn,S1,F1),STATE),
      S1 <= Start,
      F1 >= Start.

realeventconflict(ewp,Start,Finish,STATE):-
      member(event(SQ,ewn,S1,F1),STATE),
      S1 <= Finish,
      F1 >= Finish.

realeventconflict(ewn,Start,Finish,STATE):-
      member(event(SQ,ewp,S1,F1),STATE),
      S1 <= Start,
      F1 >= Start.

realeventconflict(ewn,Start,Finish,STATE):-
      member(event(SQ,ewp,S1,F1),STATE),
      S1 <= Finish,
      F1 >= Finish.

realeventconflict(_,Start,Finish,STATE):- fail.

groupsquadron(Sq,State,event(Sq,E,S,F)):-
      member(event(Sq,E,S,F),State).

groupteam(Eventname,State,event(Sq,Eventname,S,F)):-
      member(event(Sq,Eventname,S,F),State).

/*  Squadron cost .                                                    */

scost2x([X,Y|L],Cost):-
      spiece_cost(X,Y,Cost2),
      scost2x([Y|L],Cost3),
      Cost is Cost2 + Cost3.

scost2x([X],0).

scost2x([],0).

/* Team cost                                                           */

tcost2x([X,Y|L],Cost):-
      tpiece_cost(X,Y,Cost2),
      tcost2x([Y|L],Cost3),
      Cost is Cost2 + Cost3.

tcost2x([X],0).

tcost2x([],0).
```

```
tpiece_cost(event(Sq1,E1,S1,F1),event(Sq2,E2,S2,F2),Cost):-
      D is S2 - F1,
      costfn3(D,Cost).

/************************** Follow-on EVENT ****************************/
/* First argument occurs BEFORE next argument and second argument on or after  */
/* schedule begin date and this is used in squadron and team costs.            */

/* Cost for related events has min at delay = 21.                              */
spiece_cost(event(Sq,ewp,S1,F1),event(Sq,ewn,S2,F2),Cost):-
      D is S2 - F1 ,
      costfn1(D,Cost).

spiece_cost(event(Sq,ewc,S1,F1),event(Sq,ewm,S2,F2),Cost):-
      D is S2 - F1 ,
      costfn1(D,Cost).

/* Cost for unrelated events has min at delay = 21.                            */
spiece_cost(event(Sq1,E1,S1,F1),event(Sq2,E2,S2,F2),Cost):-
      D is S2 - F1,
      D < 180,
      costfn2(D,Cost).

/* The next rule is used when D > 180 to discount gaps caused by deployments.  */
spiece_cost(event(Sq1,E1,S1,F1),event(Sq2,E2,S2,F2),0).

/************************** FIRST EVENT ****************************/
spiece_cost(start,event(Sq,tr1a,S,F),0) .
spiece_cost(start,event(Sq,tr1b,S,F),0) .
spiece_cost(start,event(Sq,tr1c,S,F),0) .
spiece_cost(start,event(Sq,ec,S,F),0) .

spiece_cost(start,event(Sq,ewm,S,F),Cost):-
      priorevent(Sq,ewc,S1,F1),
      D is S - S1,
      costfn1(D,Cost).

spiece_cost(start,event(Sq,ewn,S,F),Cost):-
      priorevent(Sq,ewp,S1,F1),
      D is S - F1,
      costfn1(D,Cost).

spiece_cost(start,event(Sq,Event,S,F),Cost):-
      ((Event = ewp;
      Event = ewn);
      Event = en),
      prerequisite(Sq,ds0,S1,F1),
      D is S - F1,
      costfn2(D,Cost).
```

```
spiece_cost(start,event(Sq,Event,S,F),Cost):-
      (Event = ewc;
      Event = ewm),
       prerequisite(Sq,dv1,S1,F1),
      D is S - S1,
      costfn2(D,Cost).

/* Next rules are cost function for non-related events.                    */
costfn2(D1,Cost) :-
      D1 < 30,
      D is D1/21 ,
      Cost is  100 - 243*D + 189*D**2 - 45*D**3,!.

costfn2(D1,Cost) :-
      Cost is 10 - 6e-1*D1 + D1**2/100, !.

/* Next rules are cost function for  squadron related-events.             */

costfn1(D1,Cost) :-
      D1 < 21,
      D is D1/21 ,
      Cost is  100 - 243*D + 189*D**2 - 45*D**3, !.

costfn1(D1,Cost) :-
      Cost is   10 - 0.85714e0*D1 + D1*D1/49, !.

/* Next rules are teamcost function.                                       */
costfn3(D1,Cost) :-
      D1 < 51,
      D is D1/30,
      Cost is  100 -247.25e0*D + 197.5e0*D**2 - 49.25e0*D**3, !.

costfn3(D1,Cost) :-
      Cost is 8.52e0 + 1/(50/D1)**2, !.

eval2(State,Eval):-
      total_events(T),
      list_length(State,N2),
      X is T - N2,                          /* X is number events left to schedule . */
      Eval is X, +.                         /* Minimum va.'e of cost is 1.0.         */

cleandatabase2 :-
      cleandatabase,
      del_all_statements(index/1),
      del_all_statements(min_cost_reached/0),
      del_all_statements(oldcost/1),
      del_all_statements(beststate/2).

expon(X,Y) :-
      expon1(X,Y).
```

```
expon(X,Y) :-
      NX is -X,
      expon1(NX,Y1),
      Y is 1/Y1.

expon1(X,Y) :-
      expon2(X,1,1,1,Y).

expon2(X,N,S,T,S) :-
      T<1.0E-3,+.

expon2(X,N,S,T,Y) :-
      TP1 is X/N*T,
      SP1 is S+TP1,
      NP1 is N+1,
       expon2(X,NP1,SP1,TP1,Y) .

endmod  /* module vpfinalsearch */ .
```

VPUTILITIES MODULE

This module contains the numerous utility functions used in several other modules. It was written by Professor Rowe and LCDR Hutson in C-Prolog, and has been converted to run in M-Prolog.

```
module vputilities.

export (append / 3, delete / 3, singlemember / 2, deleteitems / 3,first / 2, member / 2,
    open / 2, tell / 2, told / 1, deleteone / 3, stars / 0, subset /2, prettyprint / 1, max / 2,
     union / 3, xbagof / 3, count1 / 1, count2 / 1, count3 / 1, counter1 /1, sortevent / 2,
    sortmonthevent / 2, abs / 2, myname /2).

global (status,trialperiod,prerequisite,event,
      jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,
      monday,tuesday,wednesday,thursday,friday,saturday,sunday,weekend,
       tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
      vp9,vp19,vp40,vp46,vp47,vp48,vp50).

/*$eject*/
body.

dynamic(count1/1).
dynamic(count2/1).
dynamic(count3/1).

abs(X,X) :- X >= 0.

aos(X,Y) :- Y is 0 - X.

last([X],X).

last([X|L],Y) :- last(L,Y).

member(X,[X|L]).

member(X,[Y|L]) :- member(X,L).

max([X],X).

max([X|L],X) :- max(L,M),X>M.

max([X|L],M) :-
     max(L,M).

min([X],X).

min([X|L],X) :- min(L,M), X<M.
```

```
min([X|L],M) :- min(L,M), X>=M.

delete(X,[],[]).

delete(X,[X|L],M) :- !, delete(X,L,M).

delete(X,[Y|L],[Y|M]) :- delete(X,L,M).

append([],L,L).

append([X|L],L2,[X|L3]) :- append(L,L2,L3).

/* Some variants.                                                    */
singlemember(X,[X|L]) :- !.

singlemember(X,[Y|L]) :- singlemember(X,L).

deleteone(X,[],[]).

deleteone(X,[X|L],L) :- !.

deleteone(X,[Y|L],[Y|M]) :- deleteone(X,L,M).

/* Writes 60 stars.                                                  */
stars :-
    write("************************************************************"),

/* Predicates defined from others,                                   */
subset([],L).

subset([X|L],L2) :- singlemember(X,L2), subset(L,L2).

/* Prints out a list with one item per line; useful for lists of lists  which can overflow  */
/* the terminal line.                                                                        */
prettyprint([]) :- nl,!.

prettyprint([X|L]) :- write(X), nl, prettyprint(L).

union([],L,L).

union(L,[],L).

union([X|L1],L2,L3) :- singlemember(X,L2), !, union(L1,L2,L3).

union([X|L1],L2,[X|L3]) :- union(L1,L2,L3).

xbagof(X,P,L):- bag_of(X,P,L),!.

xbagof(X,P,L):- !.
```

```
/* Counter for the vpgenerator module. count1(0).is asserted in init_dynamics in the */
/* vpscheduler module.                                                               */

counter1(K):-
      count1(K0),
      del_statement(count1(K0)),
      K is K0 + 1,
      add_statement(count1(K)),!.

/* Counter for the nopathsearch. count2(0). asserted in init_dynamics in vpdcheduler */
/* module .                                                                          */

counter2(K):-
      count2(K0),
      del_statement(count2(K0)),
      K is K0 + 1,
      add_statement(count2(K)),!.

/* Counter for count_priorevents.  count3(0). is asserted in init_dynamics in        */
/* vpscheduler module.                                                               */

counter3:-
      count3(K0),
      del_statement(count3(K0)),
      K is K0 + 1,
      add_statement(count3(K)),!.


/* Sorts events by start dates.                                                      */

sortevent([],[]).

sortevent([event(Sq,E,S,F)|L1],L2):-
      sortevent(L1,L3),
      insert_event(event(Sq,E,S,F),L3,L2).

insert_event(event(Sq,E,S,F),[],[event(Sq,E,S,F)]).

insert_event(event(Sq,E,S,F),[event(Sq2,E2,S2,F2)|L],[event(Sq,E,S,F),event(Sq2,
     E2,S2,F2)|L]):-
           S < S2.

insert_event(event(Sq,E,S,F),[event(Sq2,E2,S2,F2)|L],[event(Sq2,E2,S2,F2)|L2]):-
      S >= S2,
      insert_event(event(Sq,E,S,F),L,L2).

/* Sorts month events by month number.                                               */

sortmonthevent([],[]).

sortmonthevent([event(Sq,E,MN)|L1],L2):-
      sortmonthevent(L1,L3),
      insertmonthevent(event(Sq,E,MN),L3,L2).
```

```
insertmonthevent(event(Sq,E,MN),[],[event(Sq,E,MN)]).

insertmonthevent(event(Sq,E,MN),[event(Sq2,E2,MN2)|L],[event(Sq,E,MN),event(Sq2,
    E2,MN2)|L]):-
        MN < MN2.

insertmonthevent(event(Sq,E,MN),[event(Sq2,E2,MN2)|L],[event(Sq2,E2,MN2)|L2]):-
    MN >= MN2,
    insertmonthevent(event(Sq,E,MN),L,L2).

open(CH,OUTFILE):-
    set_channel(outfile(CH),[name=OUTFILE,mode=create]),
    set_output(outfile(CH)),
    told(CH).

tell(CH,OUTFILE):-
    set_channel(outfile(CH),[name=OUTFILE,mode=append]),
    set_output(outfile(CH)).

told(CH) :-
    close_output(outfile(CH)).

/* Deletes a set of items from another list .                              */
deleteitems([],L,L).

deleteitems([X|L],L2,L3) :- delete(X,L2,L4), deleteitems(L,L4,L3).

first([X|L],X).

/*  myname developed to use in the compiled version  where name is not allowed.  */
myname(S,L):-
    S is_a var,x1name(S,L).

myname(S,L):-
    L is_a var,x2name(S,L).

x1name(X,[F]) :-
   convert_char(FS,F), concatenate("",FS,X) .

x1name(X,[F|L]) :-
   convert_char(FS,F), x1name(X1,L), concatenate(FS,X1,X) .

x2name(S,[N]) :-
   string_length(S,1), convert_char(S,N) .

x2name(S,[N|NL]) :-
   string_length(S,LS), substring(S,1,1,S1), convert_char(S1,N),
     substring(S,2,LS,S2), x2name(S2,NL) .

endmod /* module vputilities */ .
```

VPSCHEDWRITER MODULE

This module contains the functions necessary to print the schedule information, in columnar format, to the screen and the file SCHEDULE.pro .

```
module vpschedwriter .

export( eventprinter / 1).

import( number_to_month / 3, eventcode / 2, daynumber_to_date / 2, open / 2,  tell / 2,
    told / 1).

global (status,trialperiod,prerequisite,event,
     ntpi,pre_mrci,mrci,natops,command,ready_alert,pre_ntpi,
      jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,
      monday,tuesday,wednesday ,hursday,friday,saturday,sunday,weekend,
       tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
      vp9,vp19,vp40,vp46,vp47,vp48,vp50).

/*$eject*/
body.

/* Writes in columnar form the schedule given a list with month numbers.          */
eventprinter(LIST) :-
     eventprinter2(LIST),!.

eventprinter2([X|L]) :-
      structure(X,list,[event,Sq,INSP,Start,Finish]),
     write(Sq),write_tab(6),
     eventcode(INSP,EVTNAME),
     write(EVTNAME),write_tab(18),
     daynumber_to_date(Start,[SD,SM,SY]),
     write(SD),write_tab(21),
     write(SM),write_tab(25),
     write(SY),write_tab(30),
     daynumber_to_date(Finish,[FD,FM,FY]),
     write(FD),write_tab(33),
     write(FM),write_tab(37),
     write(FY), nl,
     eventprinter2(L),!.

eventprinter2(X) :-
     write("Input to eventprinter was in wrong format."),nl,
      write(X),nl,!.

endmod  /* module vpschedwriter */ .
```

VPGENERATOR MODULE

This module generates all the possible trialperiods for each of the events for the schedule period. It was written by LCDR Hutson in C-Prolog and has been converted to run in M-prolog

```
module vpgenerator.

export (trialperiod / 3, generate_trialperiods / 0).

import (eventnames / 1, member / 2, yearbegin / 1, holiday / 3, daynumber_to_date / 2,
    daysofmonth / 2, counter1 / 1, count1 / 1, date_calc / 3, day_of_week /2, yearend / 1).

global (jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,
    status,trialperiod,prerequisite,
      tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
     vp9,vp19,vp40,vp46,vp47,vp48,vp50,
     monday,tuesday,wednesday,thursday,friday,saturday,sunday,weekend).

/*$eject*/
body.

dynamic(trialperiod/3).

/* Driver for this program.                                          */

generate_trialperiods:-
    write("Generating possible inspection periods...."),nl,fail.

generate_trialperiods:-
    eventnames(EN),
    member(X,EN),
    generate_readies(X),
    generate_weapevals(X),
    generate_natops(X),
    generate_ore_weapevals(X),
    generate_ore_command(X), ! .
```

```
/* The ready-alert periods are generated.                                  */
generate_readies(Eventname):-
     Eventname = tr1a,
     yearbegin(B),
     calc_start(B,0,S),
     daynumber_to_date(S,[Day,Month,Year]),
     Day = 1,
     daysofmonth(Month,D),
     calc_finish(S,D,F),
     add_statement(trialperiod(tr1,S,F),bottom),
     counter1(K),
     fail .

generate_readies(Eventname):-
      trialperiod(tr1,_,_),!.

/* Pre-NTPI and NTPI trialperiods.                                         */
generate_weapevals(Eventname):-
     (Eventname = ewp;
     Eventname = ewn),
     yearbegin(B),
     next_safedaynumber(2,B,0,S),
     calc_finish(S,2,F),
     safeperiod(S,F),
     add_statement(trialperiod(Eventname,S,F),bottom),
     counter1(K),
     fail .

generate_weapevals(Eventname):-
      trialperiod(ewp,_,_),!,
      trialperiod(ewn,_,_),!.

/* NATOPS periods are generated.                                           */
generate_natops(Eventname):-
     Eventname = en,
     yearbegin(B),
     next_safemonday_daynum(B,0,S),
     calc_finish(S,12,F),
     safeperiod_withweekends(S,F),
     add_statement(trialperiod(en,S,F),bottom),
     counter1(K),
     fail .

generate_natops(Eventname):-
      trialperiod(en,_,_),!.

/* pre_MRCI and MRCI periods.                                              */
generate_ore_weapevals(Eventname):-
     (Eventname = ewc;
```

```
      Eventname = ewm),
      yearbegin(B),
      next_safedaynumber(4,B,0,S),
      calc_finish(S,4,F),
      safeperiod(S,F),
      add_statement(trialperiod(Eventname,S,F),bottom),
      counter1(K),
      fail .

generate_ore_weapevals(Eventname):-
       trialperiod(ewc,_,_),!,
       trialperiod(ewm,_,_),!.

/* CI periods are generated.                                          */

generate_ore_command(Eventname):-
      yearbegin(B),
      next_safemonday_daynum(B,0,S),
      calc_finish(S,5,F),
      safeperiod(S,F),
      add_statement(trialperiod(ec,S,F),bottom),
      counter1(K),
      fail .

/*  Produces a final count of the number of trialperiods generated.   */

generate_ore_command(Eventname):-
       trialperiod(ec,_,_),!,
       count1(N),
       write(N),
       del_statement(count1(N)),
       add_statement(count1(0)),
       write("  Trial Periods Generated."),nl,!.

/*  Based on duration of the inspection outputs the next date, discarding holidays  */
/*  and weekends.                                                                   */

next_safedaynumber(Duration,DayNumberIn,Delay,DayNumberOut):-
      Duration = 2,
      date_calc(DayNumberIn,Delay,DayNumberOut),
      day_of_week(DayNumberOut,Day),
      not(Day = friday),
      yearend(End),
      DayNumberOut < End,
      safedaynumber(DayNumberOut).
```

```
next_safedaynumber(Duration,DayNumberIn,Delay,DayNumberOut):-
      Duration = 4,
      date_calc(DayNumberIn,Delay,DayNumberOut),
      day_of_week(DayNumberOut,Day),
      (Day = monday;
      Day = tuesday),
      yearend(End),
      DayNumberOut < End,
      safedaynumber(DayNumberOut).

next_safedaynumber(Duration,DayNumberIn,Delay,DayNumberOut):-
      Delay2 is Delay + 1,
      P is DayNumberIn + Delay2,
      yearend(End),
      P < End,
      next_safedaynumber(Duration,DayNumberIn,Delay2,DayNumberOut).

/*  Computes a non-holiday monday to start an inspection.                    */

next_safemonday_daynum(DayNumberIn,Delay,DayNumberOut):-
      date_calc(DayNumberIn,Delay,DayNumberOut),
      yearend(End),
      DayNumberOut < End,
      safemonday(DayNumberOut).

next_safemonday_daynum(DayNumberIn,Delay,DayNumberOut):-
      Delay2 is Delay + 1,
      date_calc(DayNumberIn,Delay2,ProspectiveDayNumber),
      yearend(End),
      ProspectiveDayNumber < End,
      next_safemonday_daynum(DayNumberIn,Delay2,DayNumberOut).

/*  Verifies monday is safe.                                                 */

safemonday(DayNumber):-
      day_of_week(DayNumber,Day),
      Day = monday,
      not(holiday(DayNumber,Day,Holiday_name)).

/*  Verifies inspection period is not interrupted by a weekend or holiday.   */

safeperiod(Start,Finish):-
      Start = Finish,!.

safeperiod(Start,Finish):-
      D is Start + 1,
      safedaynumber(D),
      safeperiod(D,Finish).
```

```
/* Permits exclusion of weekends with holidays since impratical to schedule a two    */
/*  week event over a holiday period anyway.                                         */
safeperiod_withweekends(Start,Finish):-
      Start = Finish,!.

safeperiod_withweekends(Start,Finish):-
      D is Start + 1,
      day_of_week(D,Day),
      not(holiday(D,Day,Holiday_name)),
      safeperiod_withweekends(D,Finish).

safedaynumber(DayNumber):-
      day_of_week(DayNumber,Day),
      not(weekend(Day)),
      not(holiday(DayNumber,Day,Holiday_name)),

/* Calculates start of event regardless of weekends or holidays.                    */
calc_start(BasedayNumber,Delay,Start):-
      Start is BasedayNumber + Delay,
      yearend(End),
      Start < End + 31.

calc_start(BasedayNumber,Delay,Start):-
      Delay2 is Delay + 1,
      P is BasedayNumber + Delay2,
      yearend(End),
      P < End + 31,                  /* Provides ready in case overlap with nextyear */
      calc_start(BasedayNumber,Delay2,Start).

/* Calculates finish of event.                                                      */
calc_finish(Start,Duration,Finish):-
      Days is Duration - 1,
      date_calc(Start,Days,Finish).

endmod  /* module vpgenerator */.
```

VPCALENDAR MODULE

This module contains all the various calendar functions to convert from one form of representing a date to another. It was written by LCDR Hutson in C-prolog and has been converted to run in M-Prolog. A new rule was added to daynumbertodate to handle leap year correctly.

```
module vpcalendar.

import (abs / 2).

export (day_of_week /2, holiday / 3, difference_between_dates2 / 3, datetodaynumber / 2,
     daysofmonth /2, date_calc / 3, daynumber_to_date / 2 ).

global (status,trialperiod,prerequisite,
      jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec,
     daysofmonth /2, date_calc / 3, daynumber_to_date / 2 ).
      monday,tuesday,wednesday,thursday,friday,saturday,sunday,weekend,
       tr1a,tr1b,tr1c,ewp,ewn,en,ewc,ewm,ec,dv1,dr1,ds1,tr1,ds0,
      vp9,vp19,vp40,vp46,vp47,vp48,vp50).

/*$eject*/
body.

/*  Determines day of week                                                    */

day_of_week(DayNumber,Day):-
     X is DayNumber mod 7,
     daymod(Day,X).

/*  Computes the number of days difference between two dates.                 */

difference_between_dates(Date1,Date2,Difference):-
     datetodaynumber(Date1,Daynumber1),
     datetodaynumber(Date2,Daynumber2),
     X is Daynumber1 - Daynumber2,
     abs(X,Difference).

difference_between_dates2(DayNumber1,DayNumber2,Difference):-
     X is DayNumber1 - DayNumber2,
     abs(X,Difference).
```

```
/* Uses 1600 as base date for ease of Gregorian correction.                    */
datetodaynumber([Monthday,Month,Year],Daynumber):-
     Diff is Year - 1600,
     N is (Diff*365)+(Diff div 4)+(Diff div 400)-(Diff div 100) + 1,
     days_so_far([Monthday,Month,Year],Days),
     Daynumber is N + Days.

days_so_far([Monthday,Month,Year],Days):-
     leapyear(Year),
     (Month = jan ; Month = feb),
     daysuntilmonth(Month,Days1),!,
     Days is Days1 + Monthday - 1 .

days_so_far([Monthday,Month,Year],Days):-
     leapyear(Year),
     not((Month = jan ; Month = feb)),
     daysuntilmonth(Month,Days1),!,
     Days is Days1 + Monthday  .

days_so_far([Monthday,Month,Year],Days):-
     daysuntilmonth(Month,Days1),!,
     Days is Days1 + Monthday.

leapyear(Year):-
     X is Year mod 400,
     X = 0,!.

leapyear(Year):-
     X is Year mod 100,
     not(X = 0),
     Y is Year mod 4, Y = 0,!.

/*        Computes  date after adding a positive or negative number of days   */
date_calc(DayNumberIn,Days,DayNumberOut):-
     DayNumberOut is Days + DayNumberIn,!.

date_calc2(DateIn,Days,DateOut):-
     datetodaynumber(DateIn,Daynumber),
     Daycount is Days + Daynumber,
     daynumber_to_date(Daycount,DateOut),!.
```

```
/* receives daynumber representing the nth day since 01 Jan 1600 and returns a date.   */
daynumber_to_date(DayCount,[Monthday,W,Year]):-
      Year is 1600 + (DayCount div 365),
      Diff is Year - 1600,
      N is (Diff*365)+(Diff div 4)+(Diff div 400)-(Diff div 100) + 1,
      not(N >= DayCount),
      not(leapyear(Year)),
      Days is DayCount - N,
      daysuntilmonth(X,Y),
      Y >= Days,
      premonth(W,X),!,
      daysuntilmonth(W,Z),
      Monthday is Days - Z,!.

daynumber_to_date(DayCount,[Monthday,W,Year]):-
      Year is 1600 + (DayCount div 365),
      Diff is Year - 1600,
      N is (Diff*365)+(Diff div 4)+(Diff div 400)-(Diff div 100) + 1,
      not(N > DayCount),
      leapyear(Year),
      Days is DayCount - N + 1,
      Days > 60,
      daysuntilmonth(X,Y),
      Y >= Days,
      premonth(W,X),!,
      daysuntilmonth(W,Z),
      Monthday is Days - Z - 1,!.

/* Handles the leap year jan and feb conversions,                                    */
daynumber_to_date(DayCount,[Monthday,W,Year]):-
      Year is 1600 + (DayCount div 365),
      Diff is Year - 1600,
      N is (Diff*365)+(Diff div 4)+(Diff div 400)-(Diff div 100) + 1,
      not(N > DayCount),
      leapyear(Year),
      Days is DayCount - N + 1,
      lpyrdaysuntilmonth(X,Y),
      Y >= Days,
      premonth(W,X),!,
      daysuntilmonth(W,Z),
      Monthday is Days - Z ,!.
```

```
daynumber_to_date(DayCount,[Monthday,W,Year]):-
    Year1 is 1600 + (DayCount div 365),
    Diff is Year1 - 1600.
    N is (Diff*365)+(Diff div 4)+(Diff div 400)-(Diff div 100) + 1,
    N >= DayCount,
    Year is Year1 - 1,
    Diff2 is Year - 1600,
    N2 is (Diff2*365)+(Diff2 div 4)+(Diff2 div 400)-(Diff2 div 100)+
    not(leapyear(Year)),
    Days is DayCount - N2,
    daysuntilmonth(X,Y),
    Y >= Days,
    premonth(W,X),!,
    daysuntilmonth(W,Z),
    Monthday is Days - Z,!.

daynumber_to_date(DayCount,[Monthday,W,Year]):-
    Year1 is 1600 + (DayCount div 365),
    Diff is Year1 - 1600,
    N is (Diff*365)+(Diff div 4)+(Diff div 400)-(Diff div 100) + 1,
    N >= DayCount,
    Year is Year1 - 1,
    Diff2 is Year - 1600,
    N2 is (Diff2*365)+(Diff2 div 4)+(Diff2 div 400)-(Diff2 div 100)+
    leapyear(Year),
    Days is DayCount - N2 + 1,
    daysuntilmonth(X,Y),
    Y >= Days-1,
    premonth(W,X),!,
    daysuntilmonth(W,Z),
    Monthday is Days - Z - 1,!.

holiday(Daynumber,Day,Holiday_name):-
    daynumber_to_date(Daynumber,Date),
    holiday_day(Date,Day,Holiday_name),! .

holiday_day([Date_day,Month,Year],Day,Holiday_name):-
    not(holiday_free_month(Month)),
    Day = monday,
    which_monday(Date_day,Monday_position),
    monday_holiday(Monday_position,Month,Holiday_name),! .

holiday_day([Date_day,Month,Year],Day,memorial_day):-
    Month = may,
    Day = monday,
    Date_day > 24.                                  /* Last Monday in May . */

holiday_day([Date_day,Month,Year],Day,thanksgiving):-
    Month = nov,
    Day = thursday,
    Date_day > 21 .
```

```
holiday_date([Date_day,Month,Year],Holiday_name):-
     not(holiday_free_month(Month)),
     day_of_week([Date_day,Month,Year],Day),
     Day = monday,
     which_monday(Date_day,Monday_position),
     monday_holiday(Monday_position,Month,Holiday_name),! .

holiday_date([Date_day,Month,Year],memorial_day):-
     Month = may,
     day_of_week([Date_day,Month,Year],Day),
     Day = monday,
     Date_day > 24, ! .

holiday_date([Date_day,Month,Year],thanksgiving):-
     Month = nov,
     day_of_week([Date_day,Month,Year],Day),
     Day = thursday,
     Date_day > 21,! .

/*     Holidays                                                    */

holiday_date([1,jan,Year],new_years).
holiday_date([4,jul,Year],independence_day).
holiday_date([11,nov,Year],veterans_day).
holiday_date([25,dec,Year],christmas).
holiday_day([4,jul,Year],Day,independence_day):- !.
holiday_day([1,jan,Year],Day,new_years) :- !.
holiday_day([11,nov,Year],Day,veterans_day):- !.
holiday_day([25,dec,Year],Day,christmas):- !.

which_monday(Date_day,Monday_position):-
     Date_day < 8, Monday_position is 1.

which_monday(Date_day,Monday_position):-
     Date_day > 7, Date_day <  15,  Monday_position is 2.

which_monday(Date_day,Monday_position):-
     Date_day > 14,Date_day <  22,  Monday_position is 3.

which_monday(Date_day,Monday_position):-
     Date_day > 21,Date_day <  29,  Monday_position is 4.

which_monday(Date_day,Monday_position):-
     Date_day > 28,Date_day <= 31,  Monday_position is 5.

/*  check to see if day is part of weekend                      */

weekend(saturday).

weekend(sunday).
```

```
monday_holiday(3,jan,martin_luther_king) .
monday_holiday(3,feb,washingtons_bday).
monday_holiday(1,sep,labor_day).
monday_holiday(2,oct,columbus_day).

daysofmonth(jan,31).
daysofmonth(feb,28).
daysofmonth(mar,31).
daysofmonth(apr,30).
daysofmonth(may,31).
daysofmonth(jun,30).
daysofmonth(jul,31).
daysofmonth(aug,31).
daysofmonth(sep,30).
daysofmonth(oct,31).
daysofmonth(nov,30).
daysofmonth(dec,31).

daymod(sunday,2).
daymod(monday,3).
daymod(tuesday,4).
daymod(wednesday,5).
daymod(thursday,6).
daymod(friday,0).
daymod(saturday,1).

premonth(jan,feb).
premonth(feb,mar).
premonth(mar,apr).
premonth(apr,may).
premonth(may,jun).
premonth(jun,jul).
premonth(jul,aug).
premonth(aug,sep).
premonth(sep,oct).
premonth(oct,nov).
premonth(nov,dec).
premonth(dec,jan).

daysuntilmonth(jan,0) .
daysuntilmonth(feb,31) .
daysuntilmonth(mar,59) .
daysuntilmonth(apr,90) .
daysuntilmonth(may,120) .
daysuntilmonth(jun,151) .
daysuntilmonth(jul,181) .
daysuntilmonth(aug,212) .
daysuntilmonth(sep,243) .
```

```
daysuntilmonth(oct,273) .
daysuntilmonth(nov,304) .
daysuntilmonth(dec,334) .
daysuntilmonth(jan,365) .                    /* added if premonth is dec */

/* Help to handle leap year jan and feb.                                 */

lpyrdaysuntilmonth(jan,0) .
lpyrdaysuntilmonth(feb,31) .
lpyrdaysuntilmonth(mar,60) .
lpyrdaysuntilmonth(apr,91) .

/* HOLIDAYS                                                              */

holiday_free_month(mar) .
holiday_free_month(apr) .
holiday_free_month(jun) .
holiday_free_month(aug) .

endmod /* module vpcalendar */.
```

APPENDIX B - SAMPLE INPUT DATA FILE

The following is an example of what a complete Database.pro file contains after the vpinterface module is completed. The information in this same database file could be modified to make subsequent runs. If it was not modified, the program would run without computing any of the information during stage I, because this is a complete database.

DATABASE.pro File

```
yearbegindate([1,oct,1985])
yearenddate([30,sep,1986])

prioreventdate(vp9,ewp,[11,apr,1985],[28,feb,1985])
prioreventdate(vp40,ewp,[11,apr,1985],[28,oct,1984])
prioreventdate(vp47,ewp,[11,apr,1985],[28,aug,1985])
prioreventdate(vp50,ewp,[11,apr,1985],[28,feb,1985])
prioreventdate(vp9,ewn,[11,apr,1985],[28,mar,1985])
prioreventdate(vp40,ewn,[11,apr,1985],[28,nov,1984])
prioreventdate(vp46,ewn,[11,apr,1985],[28,oct,1984])
prioreventdate(vp47,ewn,[11,apr,1985],[28,sep,1985])
prioreventdate(vp50,ewn,[11,apr,1985],[28,mar,1985])
prioreventdate(vp9,en,[23,jul,1985],[30,apr,1985])
prioreventdate(vp19,en,[23,jul,1985],[30,sep,1985])
prioreventdate(vp40,en,[23,jul,1985],[30,jan,1985])
prioreventdate(vp46,en,[23,jul,1985],[30,dec,1984])
prioreventdate(vp47,en,[23,jul,1985],[30,aug,1985])
prioreventdate(vp48,en,[23,jul,1985],[30,jul,1985])
prioreventdate(vp9,ewc,[15,aug,1985],[19,jan,1985])
prioreventdate(vp9,ewm,[19,sep,1985],[23,feb,1985])
prioreventdate(vp19,tr1a,[01,sep,1985],[15,oct,1985])
prioreventdate(vp40,ewc,[15,aug,1985],[19,nov,1984])
prioreventdate(vp40,ewm,[19,sep,1985],[23,dec,1984])
prioreventdate(vp46,ewc,[15,aug,1985],[19,sep,1984])
prioreventdate(vp46,ewm,[19,sep,1985],[23,oct,1984])
prioreventdate(vp47,ewm,[01,sep,1985],[25,apr,1984])
prioreventdate(vp50,ewc,[01,aug,1985],[25,feb,1985])
prioreventdate(vp50,ewm,[01,sep,1985],[25,mar,1985])
prerequisitedate(vp9,ds0,[11,dec,1985],[10,jan,1986])
prerequisitedate(vp9,dv1,[01,jul,1986],[30,sep,1986])
prerequisitedate(vp9,dr1,[16,nov,1986],[05,may,1987])
prerequisitedate(vp9,ds1,[06,may,1987],[05,apr,1987])
prerequisitedate(vp19,ds0,[11,mar,1985],[10,apr,1985])
prerequisitedate(vp19,dv1,[15,aug,1985],[15,    1985])
prerequisitedate(vp19,dr1,[10,feb,1986],[10,aug,1986])
prerequisitedate(vp19,ds1,[11,aug,1986],[10,sep,1986])
```

prerequisitedate(vp40,ds0,[11,sep,1985],[10,oct,1985])
prerequisitedate(vp40,dv1,[01,apr,1986],[30,jun,1986])
prerequisitedate(vp40,dr1,[10,aug,1986],[10,feb,1987])
prerequisitedate(vp40,ds1,[11,feb,1987],[10,mar,1987])
prerequisitedate(vp46,ds0,[11,aug,1985],[10,sep,1985])
prerequisitedate(vp46,dv1,[01,mar,1986],[31,may,1986])
prerequisitedate(vp46,dr1,[10,jul,1986],[10,jan,1987])
prerequisitedate(vp46,ds1,[11,jan,1987],[10,feb,1987])
prerequisitedate(vp47,ds0,[11,feb,1985],[10,mar,1985])
prerequisitedate(vp47,dv1,[01,sep,1985],[30,nov,1985])
prerequisitedate(vp47,dr1,[10,jan,1986],[10,jul,1986])
prerequisitedate(vp47,ds1,[11,jul,1986],[10,aug,1986])
prerequisitedate(vp48,ds0,[11,jan,1985],[10,feb,1985])
prerequisitedate(vp48,dv1,[01,aug,1985],[31,oct,1985])
prerequisitedate(vp48,dr1,[10,dec,1985],[10,jun,1986])
prerequisitedate(vp48,ds1,[11,jun,1986],[10,jul,1986])
prerequisitedate(vp50,ds0,[11,jan,1986],[10,feb,1986])
prerequisitedate(vp50,dv1,[01,aug,1986],[31,oct,1986])
prerequisitedate(vp50,dr1,[10,dec,1986],[10,jun,1987])
prerequisitedate(vp50,ds1,[11,jun,1987],[10,jul,1987])
earmark(vp9,ewn,[28,mar,1986])
earmark(vp40,ewn,[28,nov,1985])
earmark(vp46,ewn,[28,oct,1985])
earmark(vp47,ewn,[28,sep,1986])
earmark(vp50,ewn,[28,mar,1986])
earmark(vp9,en,[30,apr,1986])
earmark(vp19,en,[30,sep,1986])
earmark(vp40,en,[30,jan,1986])
earmark(vp46,en,[30,dec,1985])
earmark(vp47,en,[30,aug,1986])
earmark(vp48,en,[30,jul,1986])
earmark(vp50,en,[30,may,1986])
earmark(vp9,ewc,[19,jul,1986])
earmark(vp9,ewm,[23,aug,1986])
earmark(vp40,ewc,[19,may,1986])
earmark(vp40,ewm,[23,jun,1986])
earmark(vp46,ewc,[19,mar,1986])
earmark(vp46,ewm,[23,apr,1986])
earmark(vp47,ewm,[25,oct,1985])
earmark(vp50,ewc,[25,aug,1986])
earmark(vp50,ewn,[25,sep,1986])
trialperiod(tr1,140893,140923)
trialperiod(tr1,140924,140953)
trialperiod(tr1,140954,140984)
trialperiod(tr1,140985,141015)
trialperiod(tr1,141016,141043)
trialperiod(tr1,141044,141074)
trialperiod(tr1,141075,141104)
trialperiod(tr1,141105,141135)
trialperiod(tr1,141136,141165)
trialperiod(tr1,141166,141196)

trialperiod(tr1,141197,141227)
trialperiod(tr1,141228,141257)
trialperiod(tr1,141258,141288)
trialperiod(tr1,141289,141318)
trialperiod(tr1,141319,141349)
trialperiod(tr1,141350,141380)
trialperiod(ewp,140893,140894)
trialperiod(ewp,140894,140895)
trialperiod(ewp,140895,140896)
trialperiod(ewp,140899,140900)
trialperiod(ewp,140900,140901)
trialperiod(ewp,140901,140902)
trialperiod(ewp,140902,140903)
trialperiod(ewp,140907,140908)
trialperiod(ewp,140908,140909)
trialperiod(ewp,140909,140910)
trialperiod(ewp,140913,140914)
trialperiod(ewp,140914,140915)
trialperiod(ewp,140915,140916)
trialperiod(ewp,140916,140917)
trialperiod(ewp,140920,140921)
trialperiod(ewp,140921,140922)
trialperiod(ewp,140922,140923)
trialperiod(ewp,140923,140924)
trialperiod(ewp,140927,140928)
trialperiod(ewp,140928,140929)
trialperiod(ewp,140929,140930)
trialperiod(ewp,140930,140931)
trialperiod(ewp,140935,140936)
trialperiod(ewp,140936,140937)
trialperiod(ewp,140937,140938)
trialperiod(ewp,140941,140942)
trialperiod(ewp,140942,140943)
trialperiod(ewp,140943,140944)
trialperiod(ewp,140944,140945)
trialperiod(ewp,140948,140949)
trialperiod(ewp,140949,140950)
trialperiod(ewp,140955,140956)
trialperiod(ewp,140956,140957)
trialperiod(ewp,140957,140958)
trialperiod(ewp,140958,140959)
trialperiod(ewp,140962,140963)
trialperiod(ewp,140963,140964)
trialperiod(ewp,140964,140965)
trialperiod(ewp,140965,140966)
trialperiod(ewp,140969,140970)
trialperiod(ewp,140970,140971)
trialperiod(ewp,140971,140972)
trialperiod(ewp,140972,140973)
trialperiod(ewp,140976,140977)
trialperiod(ewp,140979,140980)

trialperiod(ewp,140983,140984)
trialperiod(ewp,140986,140987)
trialperiod(ewp,140990,140991)
trialperiod(ewp,140991,140992)
trialperiod(ewp,140992,140993)
trialperiod(ewp,140993,140994)
trialperiod(ewp,140997,140998)
trialperiod(ewp,140998,140999)
trialperiod(ewp,140999,141000)
trialperiod(ewp,141000,141001)
trialperiod(ewp,141005,141006)
trialperiod(ewp,141006,141007)
trialperiod(ewp,141007,141008)
trialperiod(ewp,141011,141012)
trialperiod(ewp,141012,141013)
trialperiod(ewp,141013,141014)
trialperiod(ewp,141014,141015)
trialperiod(ewp,141018,141019)
trialperiod(ewp,141019,141020)
trialperiod(ewp,141020,141021)
trialperiod(ewp,141021,141022)
trialperiod(ewp,141025,141026)
trialperiod(ewp,141026,141027)
trialperiod(ewp,141027,141028)
trialperiod(ewp,141028,141029)
trialperiod(ewp,141033,141034)
trialperiod(ewp,141034,141035)
trialperiod(ewp,141035,141036)
trialperiod(ewp,141039,141040)
trialperiod(ewp,141040,141041)
trialperiod(ewp,141041,141042)
trialperiod(ewp,141042,141043)
trialperiod(ewp,141046,141047)
trialperiod(ewp,141047,141048)
trialperiod(ewp,141048,141049)
trialperiod(ewp,141049,141050)
trialperiod(ewp,141053,141054)
trialperiod(ewp,141054,141055)
trialperiod(ewp,141055,141056)
trialperiod(ewp,141056,141057)
trialperiod(ewp,141060,141061)
trialperiod(ewp,141061,141062)
trialperiod(ewp,141062,141063)
trialperiod(ewp,141063,141064)
trialperiod(ewp,141067,141068)
trialperiod(ewp,141068,141069)
trialperiod(ewp,141069,141070)
trialperiod(ewp,141070,141071)
trialperiod(ewp,141074,141075)
trialperiod(ewp,141075,141076)
trialperiod(ewp,141076,141077)

trialperiod(ewp,141077,141078)
trialperiod(ewp,141081,141082)
trialperiod(ewp,141082,141083)
trialperiod(ewp,141083,141084)
trialperiod(ewp,141084,141085)
trialperiod(ewp,141088,141089)
trialperiod(ewp,141089,141090)
trialperiod(ewp,141090,141091)
trialperiod(ewp,141091,141092)
trialperiod(ewp,141095,141096)
trialperiod(ewp,141096,141097)
trialperiod(ewp,141097,141098)
trialperiod(ewp,141098,141099)
trialperiod(ewp,141102,141103)
trialperiod(ewp,141103,141104)
trialperiod(ewp,141104,141105)
trialperiod(ewp,141105,141106)
trialperiod(ewp,141109,141110)
trialperiod(ewp,141110,141111)
trialperiod(ewp,141111,141112)
trialperiod(ewp,141112,141113)
trialperiod(ewp,141116,141117)
trialperiod(ewp,141117,141118)
trialperiod(ewp,141118,141119)
trialperiod(ewp,141119,141120)
trialperiod(ewp,141123,141124)
trialperiod(ewp,141124,141125)
trialperiod(ewp,141125,141126)
trialperiod(ewp,141126,141127)
trialperiod(ewp,141131,141132)
trialperiod(ewp,141132,141133)
trialperiod(ewp,141133,141134)
trialperiod(ewp,141137,141138)
trialperiod(ewp,141138,141139)
trialperiod(ewp,141139,141140)
trialperiod(ewp,141140,141141)
trialperiod(ewp,141144,141145)
trialperiod(ewp,141145,141146)
trialperiod(ewp,141146,141147)
trialperiod(ewp,141147,141148)
trialperiod(ewp,141151,141152)
trialperiod(ewp,141152,141153)
trialperiod(ewp,141153,141154)
trialperiod(ewp,141154,141155)
trialperiod(ewp,141158,141159)
trialperiod(ewp,141159,141160)
trialperiod(ewp,141160,141161)
trialperiod(ewp,141161,141162)
trialperiod(ewp,141165,141166)
trialperiod(ewp,141166,141167)
trialperiod(ewp,141167,141168)

trialperiod(ewp,141172,141173)
trialperiod(ewp,141173,141174)
trialperiod(ewp,141174,141175)
trialperiod(ewp,141175,141176)
trialperiod(ewp,141179,141180)
trialperiod(ewp,141180,141181)
trialperiod(ewp,141181,141182)
trialperiod(ewp,141182,141183)
trialperiod(ewp,141186,141187)
trialperiod(ewp,141187,141188)
trialperiod(ewp,141188,141189)
trialperiod(ewp,141189,141190)
trialperiod(ewp,141193,141194)
trialperiod(ewp,141194,141195)
trialperiod(ewp,141195,141196)
trialperiod(ewp,141196,141197)
trialperiod(ewp,141200,141201)
trialperiod(ewp,141201,141202)
trialperiod(ewp,141202,141203)
trialperiod(ewp,141203,141204)
trialperiod(ewp,141207,141208)
trialperiod(ewp,141208,141209)
trialperiod(ewp,141209,141210)
trialperiod(ewp,141210,141211)
trialperiod(ewp,141214,141215)
trialperiod(ewp,141215,141216)
trialperiod(ewp,141216,141217)
trialperiod(ewp,141217,141218)
trialperiod(ewp,141221,141222)
trialperiod(ewp,141222,141223)
trialperiod(ewp,141223,141224)
trialperiod(ewp,141224,141225)
trialperiod(ewp,141229,141230)
trialperiod(ewp,141230,141231)
trialperiod(ewp,141231,141232)
trialperiod(ewp,141235,141236)
trialperiod(ewp,141236,141237)
trialperiod(ewp,141237,141238)
trialperiod(ewp,141238,141239)
trialperiod(ewp,141242,141243)
trialperiod(ewp,141243,141244)
trialperiod(ewp,141244,141245)
trialperiod(ewp,141245,141246)
trialperiod(ewp,141249,141250)
trialperiod(ewp,141250,141251)
trialperiod(ewp,141251,141252)
trialperiod(ewp,141252,141253)
trialperiod(ewp,141256,141257)
trialperiod(ewp,141257,141258)
trialperiod(ewp,141258,141259)
trialperiod(ewp,141259,141260)

trialperiod(ewp,141263,141264)
trialperiod(ewp,141264,141265)
trialperiod(ewp,141265,141266)
trialperiod(ewp,141266,141267)
trialperiod(ewp,141271,141272)
trialperiod(ewp,141272,141273)
trialperiod(ewp,141273,141274)
trialperiod(ewp,141277,141278)
trialperiod(ewp,141278,141279)
trialperiod(ewp,141279,141280)
trialperiod(ewp,141280,141281)
trialperiod(ewp,141284,141285)
trialperiod(ewp,141285,141286)
trialperiod(ewp,141286,141287)
trialperiod(ewp,141287,141288)
trialperiod(ewp,141291,141292)
trialperiod(ewp,141292,141293)
trialperiod(ewp,141293,141294)
trialperiod(ewp,141294,141295)
trialperiod(ewp,141300,141301)
trialperiod(ewp,141301,141302)
trialperiod(ewp,141305,141306)
trialperiod(ewp,141306,141307)
trialperiod(ewp,141307,141308)
trialperiod(ewp,141308,141309)
trialperiod(ewp,141312,141313)
trialperiod(ewp,141313,141314)
trialperiod(ewp,141319,141320)
trialperiod(ewp,141320,141321)
trialperiod(ewp,141321,141322)
trialperiod(ewp,141322,141323)
trialperiod(ewp,141326,141327)
trialperiod(ewp,141327,141328)
trialperiod(ewp,141328,141329)
trialperiod(ewp,141329,141330)
trialperiod(ewp,141333,141334)
trialperiod(ewp,141334,141335)
trialperiod(ewp,141335,141336)
trialperiod(ewp,141336,141337)
trialperiod(ewp,141340,141341)
trialperiod(ewp,141341,141342)
trialperiod(ewn,140893,140894)
trialperiod(ewn,140894,140895)
trialperiod(ewn,140895,140896)
trialperiod(ewn,140899,140900)
trialperiod(ewn,140900,140901)
trialperiod(ewn,140901,140902)
trialperiod(ewn,140902,140903)
trialperiod(ewn,140907,140908)
trialperiod(ewn,140908,140909)
trialperiod(ewn,140909,140910)

trialperiod(ewn,140913,140914)
trialperiod(ewn,140914,140915)
trialperiod(ewn,140915,140916)
trialperiod(ewn,140916,140917)
trialperiod(ewn,140920,140921)
trialperiod(ewn,140921,140922)
trialperiod(ewn,140922,140923)
trialperiod(ewn,140923,140924)
trialperiod(ewn,140927,140928)
trialperiod(ewn,140928,140929)
trialperiod(ewn,140929,140930)
trialperiod(ewn,140930,140931)
trialperiod(ewn,140935,140936)
trialperiod(ewn,140936,140937)
trialperiod(ewn,140937,140938)
trialperiod(ewn,140941,140942)
trialperiod(ewn,140942,140943)
trialperiod(ewn,140943,140944)
trialperiod(ewn,140944,140945)
trialperiod(ewn,140948,140949)
trialperiod(ewn,140949,140950)
trialperiod(ewn,140955,140956)
trialperiod(ewn,140956,140957)
trialperiod(ewn,140957,140958)
trialperiod(ewn,140958,140959)
trialperiod(ewn,140962,140963)
trialperiod(ewn,140963,140964)
trialperiod(ewn,140964,140965)
trialperiod(ewn,140965,140966)
trialperiod(ewn,140969,140970)
trialperiod(ewn,140970,140971)
trialperiod(ewn,140971,140972)
trialperiod(ewn,140972,140973)
trialperiod(ewn,140976,140977)
trialperiod(ewn,140979,140980)
trialperiod(ewn,140983,140984)
trialperiod(ewn,140986,140987)
trialperiod(ewn,140990,140991)
trialperiod(ewn,140991,140992)
trialperiod(ewn,140992,140993)
trialperiod(ewn,140993,140994)
trialperiod(ewn,140997,140998)
trialperiod(ewn,140998,140999)
trialperiod(ewn,140999,141000)
trialperiod(ewn,141000,141001)
trialperiod(ewn,141005,141006)
trialperiod(ewn,141006,141007)
trialperiod(ewn,141007,141008)
trialperiod(ewn,141011,141012)
trialperiod(ewn,141012,141013)
trialperiod(ewn,141013,141014)

trialperiod(ewn,141014,141015)
trialperiod(ewn,141018,141019)
trialperiod(ewn,141019,141020)
trialperiod(ewn,141020,141021)
trialperiod(ewn,141021,141022)
trialperiod(ewn,141025,141026)
trialperiod(ewn,141026,141027)
trialperiod(ewn,141027,141028)
trialperiod(ewn,141028,141029)
trialperiod(ewn,141033,141034)
trialperiod(ewn,141034,141035)
trialperiod(ewn,141035,141036)
trialperiod(ewn,141039,141040)
trialperiod(ewn,141040,141041)
trialperiod(ewn,141041,141042)
trialperiod(ewn,141042,141043)
trialperiod(ewn,141046,141047)
trialperiod(ewn,141047,141048)
trialperiod(ewn,141048,141049)
trialperiod(ewn,141049,141050)
trialperiod(ewn,141053,141054)
trialperiod(ewn,141054,141055)
trialperiod(ewn,141055,141056)
trialperiod(ewn,141056,141057)
trialperiod(ewn,141060,141061)
trialperiod(ewn,141061,141062)
trialperiod(ewn,141062,141063)
trialperiod(ewn,141063,141064)
trialperiod(ewn,141067,141068)
trialperiod(ewn,141068,141069)
trialperiod(ewn,141069,141070)
trialperiod(ewn,141070,141071)
trialperiod(ewn,141074,141075)
trialperiod(ewn,141075,141076)
trialperiod(ewn,141076,141077)
trialperiod(ewn,141077,141078)
trialperiod(ewn,141081,141082)
trialperiod(ewn,141082,141083)
trialperiod(ewn,141083,141084)
trialperiod(ewn,141084,141085)
trialperiod(ewn,141088,141089)
trialperiod(ewn,141089,141090)
trialperiod(ewn,141090,141091)
trialperiod(ewn,141091,141092)
trialperiod(ewn,141095,141096)
trialperiod(ewn,141096,141097)
trialperiod(ewn,141097,141098)
trialperiod(ewn,141098,141099)
trialperiod(ewn,141102,141103)
trialperiod(ewn,141103,141104)
trialperiod(ewn,141104,141105)

trialperiod(ewn,141105,141106)
trialperiod(ewn,141109,141110)
trialperiod(ewn,141110,141111)
trialperiod(ewn,141111,141112)
trialperiod(ewn,141112,141113)
trialperiod(ewn,141116,141117)
trialperiod(ewn,141117,141118)
trialperiod(ewn,141118,141119)
trialperiod(ewn,141119,141120)
trialperiod(ewn,141123,141124)
trialperiod(ewn,141124,141125)
trialperiod(ewn,141125,141126)
trialperiod(ewn,141126,141127)
trialperiod(ewn,141131,141132)
trialperiod(ewn,141132,141133)
trialperiod(ewn,141133,141134)
trialperiod(ewn,141137,141138)
trialperiod(ewn,141138,141139)
trialperiod(ewn,141139,141140)
trialperiod(ewn,141140,141141)
trialperiod(ewn,141144,141145)
trialperiod(ewn,141145,141146)
trialperiod(ewn,141146,141147)
trialperiod(ewn,141147,141148)
trialperiod(ewn,141151,141152)
trialperiod(ewn,141152,141153)
trialperiod(ewn,141153,141154)
trialperiod(ewn,141154,141155)
trialperiod(ewn,141158,141159)
trialperiod(ewn,141159,141160)
trialperiod(ewn,141160,141161)
trialperiod(ewn,141161,141162)
trialperiod(ewn,141165,141166)
trialperiod(ewn,141166,141167)
trialperiod(ewn,141167,141168)
trialperiod(ewn,141172,141173)
trialperiod(ewn,141173,141174)
trialperiod(ewn,141174,141175)
trialperiod(ewn,141175,141176)
trialperiod(ewn,141179,141180)
trialperiod(ewn,141180,141181)
trialperiod(ewn,141181,141182)
trialperiod(ewn,141182,141183)
trialperiod(ewn,141186,141187)
trialperiod(ewn,141187,141188)
trialperiod(ewn,141188,141189)
trialperiod(ewn,141189,141190)
trialperiod(ewn,141193,141194)
trialperiod(ewn,141194,141195)
trialperiod(ewn,141195,141196)
trialperiod(ewn,141196,141197)

trialperiod(ewn,141200,141201)
trialperiod(ewn,141201,141202)
trialperiod(ewn,141202,141203)
trialperiod(ewn,141203,141204)
trialperiod(ewn,141207,141208)
trialperiod(ewn,141208,141209)
trialperiod(ewn,141209,141210)
trialperiod(ewn,141210,141211)
trialperiod(ewn,141214,141215)
trialperiod(ewn,141215,141216)
trialperiod(ewn,141216,141217)
trialperiod(ewn,141217,141218)
trialperiod(ewn,141221,141222)
trialperiod(ewn,141222,141223)
trialperiod(ewn,141223,141224)
trialperiod(ewn,141224,141225)
trialperiod(ewn,141229,141230)
trialperiod(ewn,141230,141231)
trialperiod(ewn,141231,141232)
trialperiod(ewn,141235,141236)
trialperiod(ewn,141236,141237)
trialperiod(ewn,141237,141238)
trialperiod(ewn,141238,141239)
trialperiod(ewn,141242,141243)
trialperiod(ewn,141243,141244)
trialperiod(ewn,141244,141245)
trialperiod(ewn,141245,141246)
trialperiod(ewn,141249,141250)
trialperiod(ewn,141250,141251)
trialperiod(ewn,141251,141252)
trialperiod(ewn,141252,141253)
trialperiod(ewn,141256,141257)
trialperiod(ewn,141257,141258)
trialperiod(ewn,141258,141259)
trialperiod(ewn,141259,141260)
trialperiod(ewn,141263,141264)
trialperiod(ewn,141264,141265)
trialperiod(ewn,141265,141266)
trialperiod(ewn,141266,141267)
trialperiod(ewn,141271,141272)
trialperiod(ewn,141272,141273)
trialperiod(ewn,141273,141274)
trialperiod(ewn,141277,141278)
trialperiod(ewn,141278,141279)
trialperiod(ewn,141279,141280)
trialperiod(ewn,141280,141281)
trialperiod(ewn,141284,141285)
trialperiod(ewn,141285,141286)
trialperiod(ewn,141286,141287)
trialperiod(ewn,141287,141288)
trialperiod(ewn,141291,141292)

trialperiod(ewn,141292,141293)
trialperiod(ewn,141293,141294)
trialperiod(ewn,141294,141295)
trialperiod(ewn,141300,141301)
trialperiod(ewn,141301,141302)
trialperiod(ewn,141305,141306)
trialperiod(ewn,141306,141307)
trialperiod(ewn,141307,141308)
trialperiod(ewn,141308,141309)
trialperiod(ewn,141312,141313)
trialperiod(ewn,141313,141314)
trialperiod(ewn,141319,141320)
trialperiod(ewn,141320,141321)
trialperiod(ewn,141321,141322)
trialperiod(ewn,141322,141323)
trialperiod(ewn,141326,141327)
trialperiod(ewn,141327,141328)
trialperiod(ewn,141328,141329)
trialperiod(ewn,141329,141330)
trialperiod(ewn,141333,141334)
trialperiod(ewn,141334,141335)
trialperiod(ewn,141335,141336)
trialperiod(ewn,141336,141337)
trialperiod(ewn,141340,141341)
trialperiod(ewn,141341,141342)
trialperiod(en,140913,140924)
trialperiod(en,140920,140931)
trialperiod(en,140955,140966)
trialperiod(en,140962,140973)
trialperiod(en,140990,141001)
trialperiod(en,141011,141022)
trialperiod(en,141018,141029)
trialperiod(en,141039,141050)
trialperiod(en,141046,141057)
trialperiod(en,141053,141064)
trialperiod(en,141060,141071)
trialperiod(en,141067,141078)
trialperiod(en,141074,141085)
trialperiod(en,141081,141092)
trialperiod(en,141088,141099)
trialperiod(en,141095,141106)
trialperiod(en,141102,141113)
trialperiod(en,141109,141120)
trialperiod(en,141116,141127)
trialperiod(en,141137,141148)
trialperiod(en,141144,141155)
trialperiod(en,141151,141162)
trialperiod(en,141172,141183)
trialperiod(en,141179,141190)
trialperiod(en,141186,141197)
trialperiod(en,141193,141204)

trialperiod(en,141200,141211)
trialperiod(en,141207,141218)
trialperiod(en,141214,141225)
trialperiod(en,141235,141246)
trialperiod(en,141242,141253)
trialperiod(en,141249,141260)
trialperiod(en,141256,141267)
trialperiod(en,141277,141288)
trialperiod(en,141284,141295)
trialperiod(en,141319,141330)
trialperiod(en,141326,141337)
trialperiod(ewc,140893,140896)
trialperiod(ewc,140899,140902)
trialperiod(ewc,140900,140903)
trialperiod(ewc,140907,140910)
trialperiod(ewc,140913,140916)
trialperiod(ewc,140914,140917)
trialperiod(ewc,140920,140923)
trialperiod(ewc,140921,140924)
trialperiod(ewc,140927,140930)
trialperiod(ewc,140928,140931)
trialperiod(ewc,140935,140938)
trialperiod(ewc,140941,140944)
trialperiod(ewc,140942,140945)
trialperiod(ewc,140955,140958)
trialperiod(ewc,140956,140959)
trialperiod(ewc,140962,140965)
trialperiod(ewc,140963,140966)
trialperiod(ewc,140969,140972)
trialperiod(ewc,140970,140973)
trialperiod(ewc,140990,140993)
trialperiod(ewc,140991,140994)
trialperiod(ewc,140997,141000)
trialperiod(ewc,140998,141001)
trialperiod(ewc,141005,141008)
trialperiod(ewc,141011,141014)
trialperiod(ewc,141012,141015)
trialperiod(ewc,141018,141021)
trialperiod(ewc,141019,141022)
trialperiod(ewc,141025,141028)
trialperiod(ewc,141026,141029)
trialperiod(ewc,141033,141036)
trialperiod(ewc,141039,141042)
trialperiod(ewc,141040,141043)
trialperiod(ewc,141046,141049)
trialperiod(ewc,141047,141050)
trialperiod(ewc,141053,141056)
trialperiod(ewc,141054,141057)
trialperiod(ewc,141060,141063)
trialperiod(ewc,141061,141064)
trialperiod(ewc,141067,141070)

trialperiod(ewc,141068,141071)
trialperiod(ewc,141074,141077)
trialperiod(ewc,141075,141078)
trialperiod(ewc,141081,141084)
trialperiod(ewc,141082,141085)
trialperiod(ewc,141088,141091)
trialperiod(ewc,141089,141092)
trialperiod(ewc,141095,141098)
trialperiod(ewc,141096,141099)
trialperiod(ewc,141102,141105)
trialperiod(ewc,141103,141106)
trialperiod(ewc,141109,141112)
trialperiod(ewc,141110,141113)
trialperiod(ewc,141116,141119)
trialperiod(ewc,141117,141120)
trialperiod(ewc,141123,141126)
trialperiod(ewc,141124,141127)
trialperiod(ewc,141131,141134)
trialperiod(ewc,141137,141140)
trialperiod(ewc,141138,141141)
trialperiod(ewc,141144,141147)
trialperiod(ewc,141145,141148)
trialperiod(ewc,141151,141154)
trialperiod(ewc,141152,141155)
trialperiod(ewc,141158,141161)
trialperiod(ewc,141159,141162)
trialperiod(ewc,141165,141168)
trialperiod(ewc,141172,141175)
trialperiod(ewc,141173,141176)
trialperiod(ewc,141179,141182)
trialperiod(ewc,141180,141183)
trialperiod(ewc,141186,141189)
trialperiod(ewc,141187,141190)
trialperiod(ewc,141193,141196)
trialperiod(ewc,141194,141197)
trialperiod(ewc,141200,141203)
trialperiod(ewc,141201,141204)
trialperiod(ewc,141207,141210)
trialperiod(ewc,141208,141211)
trialperiod(ewc,141214,141217)
trialperiod(ewc,141215,141218)
trialperiod(ewc,141221,141224)
trialperiod(ewc,141222,141225)
trialperiod(ewc,141229,141232)
trialperiod(ewc,141235,141238)
trialperiod(ewc,141236,141239)
trialperiod(ewc,141242,141245)
trialperiod(ewc,141243,141246)
trialperiod(ewc,141249,141252)
trialperiod(ewc,141250,141253)
trialperiod(ewc,141256,141259)

trialperiod(ewc,141257,141260)
trialperiod(ewc,141263,141266)
trialperiod(ewc,141264,141267)
trialperiod(ewc,141271,141274)
trialperiod(ewc,141277,141280)
trialperiod(ewc,141278,141281)
trialperiod(ewc,141284,141287)
trialperiod(ewc,141285,141288)
trialperiod(ewc,141291,141294)
trialperiod(ewc,141292,141295)
trialperiod(ewc,141305,141308)
trialperiod(ewc,141306,141309)
trialperiod(ewc,141319,141322)
trialperiod(ewc,141320,141323)
trialperiod(ewc,141326,141329)
trialperiod(ewc,141327,141330)
trialperiod(ewc,141333,141336)
trialperiod(ewc,141334,141337)
trialperiod(ewm,140893,140896)
trialperiod(ewm,140899,140902)
trialperiod(ewm,140900,140903)
trialperiod(ewm,140907,140910)
trialperiod(ewm,140913,140916)
trialperiod(ewm,140914,140917)
trialperiod(ewm,140920,140923)
trialperiod(ewm,140921,140924)
trialperiod(ewm,140927,140930)
trialperiod(ewm,140928,140931)
trialperiod(ewm,140935,140938)
trialperiod(ewm,140941,140944)
trialperiod(ewm,140942,140945)
trialperiod(ewm,140955,140958)
trialperiod(ewm,140956,140959)
trialperiod(ewm,140962,140965)
trialperiod(ewm,140963,140966)
trialperiod(ewm,140969,140972)
trialperiod(ewm,140970,140973)
trialperiod(ewm,140990,140993)
trialperiod(ewm,140991,140994)
trialperiod(ewm,140997,141000)
trialperiod(ewm,140998,141001)
trialperiod(ewm,141005,141008)
trialperiod(ewm,141011,141014)
trialperiod(ewm,141012,141015)
trialperiod(ewm,141018,141021)
trialperiod(ewm,141019,141022)
trialperiod(ewm,141025,141028)
trialperiod(ewm,141026,141029)
trialperiod(ewm,141033,141036)
trialperiod(ewm,141039,141042)
trialperiod(ewm,141040,141043)

trialperiod(ewm,141046,141049)
trialperiod(ewm,141047,141050)
trialperiod(ewm,141053,141056)
trialperiod(ewm,141054,141057)
trialperiod(ewm,141060,141063)
trialperiod(ewm,141061,141064)
trialperiod(ewm,141067,141070)
trialperiod(ewm,141068,141071)
trialperiod(ewm,141074,141077)
trialperiod(ewm,141075,141078)
trialperiod(ewm,141081,141084)
trialperiod(ewm,141082,141085)
trialperiod(ewm,141088,141091)
trialperiod(ewm,141089,141092)
trialperiod(ewm,141095,141098)
trialperiod(ewm,141096,141099)
trialperiod(ewm,141102,141105)
trialperiod(ewm,141103,141106)
trialperiod(ewm,141109,141112)
trialperiod(ewm,141110,141113)
trialperiod(ewm,141116,141119)
trialperiod(ewm,141117,141120)
trialperiod(ewm,141123,141126)
trialperiod(ewm,141124,141127)
trialperiod(ewm,141131,141134)
trialperiod(ewm,141137,141140)
trialperiod(ewm,141138,141141)
trialperiod(ewm,141144,141147)
trialperiod(ewm,141145,141148)
trialperiod(ewm,141151,141154)
trialperiod(ewm,141152,141155)
trialperiod(ewm,141158,141161)
trialperiod(ewm,141159,141162)
trialperiod(ewm,141165,141168)
trialperiod(ewm,141172,141175)
trialperiod(ewm,141173,141176)
trialperiod(ewm,141179,141182)
trialperiod(ewm,141180,141183)
trialperiod(ewm,141186,141189)
trialperiod(ewm,141187,141190)
trialperiod(ewm,141193,141196)
trialperiod(ewm,141194,141197)
trialperiod(ewm,141200,141203)
trialperiod(ewm,141201,141204)
trialperiod(ewm,141207,141210)
trialperiod(ewm,141208,141211)
trialperiod(ewm,141214,141217)
trialperiod(ewm,141215,141218)
trialperiod(ewm,141221,141224)
trialperiod(ewm,141222,141225)
trialperiod(ewm,141229,141232)

trialperiod(ewm,141235,141238)
trialperiod(ewm,141236,141239)
trialperiod(ewm,141242,141245)
trialperiod(ewm,141243,141246)
trialperiod(ewm,141249,141252)
trialperiod(ewm,141250,141253)
trialperiod(ewm,141256,141259)
trialperiod(ewm,141257,141260)
trialperiod(ewm,141263,141266)
trialperiod(ewm,141264,141267)
trialperiod(ewm,141271,141274)
trialperiod(ewm,141277,141280)
trialperiod(ewm,141278,141281)
trialperiod(ewm,141284,141287)
trialperiod(ewm,141285,141288)
trialperiod(ewm,141291,141294)
trialperiod(ewm,141292,141295)
trialperiod(ewm,141305,141308)
trialperiod(ewm,141306,141309)
trialperiod(ewm,141319,141322)
trialperiod(ewm,141320,141323)
trialperiod(ewm,141326,141329)
trialperiod(ewm,141327,141330)
trialperiod(ewm,141333,141336)
trialperiod(ewm,141334,141337)
trialperiod(ec,140899,140903)
trialperiod(ec,140913,140917)
trialperiod(ec,140920,140924)
trialperiod(ec,140927,140931)
trialperiod(ec,140941,140945)
trialperiod(ec,140955,140959)
trialperiod(ec,140962,140966)
trialperiod(ec,140969,140973)
trialperiod(ec,140990,140994)
trialperiod(ec,140997,141001)
trialperiod(ec,141011,141015)
trialperiod(ec,141018,141022)
trialperiod(ec,141025,141029)
trialperiod(ec,141039,141043)
trialperiod(ec,141046,141050)
trialperiod(ec,141053,141057)
trialperiod(ec,141060,141064)
trialperiod(ec,141067,141071)
trialperiod(ec,141074,141078)
trialperiod(ec,141081,141085)
trialperiod(ec,141088,141092)
trialperiod(ec,141095,141099)
trialperiod(ec,141102,141106)
trialperiod(ec,141109,141113)
trialperiod(ec,141116,141120)
trialperiod(ec,141123,141127)

trialperiod(ec,141137,141141)
trialperiod(ec,141144,141148)
trialperiod(ec,141151,141155)
trialperiod(ec,141158,141162)
trialperiod(ec,141172,141176)
trialperiod(ec,141179,141183)
trialperiod(ec,141186,141190)
trialperiod(ec,141193,141197)
trialperiod(ec,141200,141204)
trialperiod(ec,141207,141211)
trialperiod(ec,141214,141218)
trialperiod(ec,141221,141225)
trialperiod(ec,141235,141239)
trialperiod(ec,141242,141246)
trialperiod(ec,141249,141253)
trialperiod(ec,141256,141260)
trialperiod(ec,141263,141267)
trialperiod(ec,141277,141281)
trialperiod(ec,141284,141288)
trialperiod(ec,141291,141295)
trialperiod(ec,141305,141309)
trialperiod(ec,141319,141323)
trialperiod(ec,141326,141330)
trialperiod(ec,141333,141337)

readyeventdate(vp47,tr1a,[1,sep,1986],[30,sep,1986])
readyeventdate(vp48,tr1a,[1,aug,1986],[31,aug,1986])
readyeventdate(vp9,tr1c,[1,jul,1986],[31,jul,1986])
readyeventdate(vp50,tr1b,[1,jun,1986],[30,jun,1986])
readyeventdate(vp9,tr1b,[1,may,1986],[31,may,1986])
readyeventdate(vp50,tr1a,[1,apr,1986],[30,apr,1986])
readyeventdate(vp9,tr1a,[1,mar,1986],[31,mar,1986])
readyeventdate(vp40,tr1b,[1,feb,1986],[28,feb,1986])
readyeventdate(vp46,tr1b,[1,jan,1986],[31,jan,1986])
readyeventdate(vp40,tr1a,[1,dec,1985],[31,dec,1985])
readyeventdate(vp46,tr1a,[16,oct,1985],[30,nov,1985])
readyeventdate(vp46,tr1a,[16,oct,1985],[30,nov,1985])

APPENDIX C - EXAMPLE FINAL SCHEDULE FILE

The following is an example of the 1986 schedule produced using this program and what would be found in the file SCHEDULE.pro.

vp46	ntpi	1	oct	1985	2	oct 1985
vp47	mrci	8	oct	1985	11	oct 1985
vp46	ready_alert	16	oct	1985	30	nov 1985
vp40	pre_ntpi	17	oct	1985	18	oct 1985
vp48	command	21	oct	1985	25	oct 1985
vp40	ntpi	31	oct	1985	1	nov 1985
vp47	command	18	nov	1985	22	nov 1985
vp40	ready_alert	1	dec	1985	31	dec 1985
vp46	natops	9	dec	1985	20	dec 1985
vp19	command	16	dec	1985	20	dec 1985
vp46	ready_alert	1	jan	1986	31	jan 1986
vp40	natops	6	jan	1986	17	jan 1986
vp9	pre_ntpi	21	jan	1986	22	jan 1986
vp40	ready_alert	1	feb	1986	28	feb 1986
vp9	ntpi	10	feb	1986	11	feb 1986
vp50	pre_ntpi	19	feb	1986	20	feb 1986
vp9	ready_alert	1	mar	1986	31	mar 1986
vp50	ntpi	12	mar	1986	13	mar 1986
vp46	pre_mrci	24	mar	1986	27	mar 1986
vp50	ready_alert	1	apr	1986	30	apr 1986
vp46	mrci	7	apr	1986	10	apr 1986
vp40	pre_mrci	8	apr	1986	1	may 1986
vp9	ready_alert	1	may	1986	31	may 1986
vp50	natops	12	may	1986	23	may 1986
vp40	mrci	12	may	1986	15	may 1986
vp46	command	19	may	1986	23	may 1986
vp50	ready_alert	1	jun	1986	30	jun 1986
vp40	command	23	jun	1986	27	jun 1986
vp47	pre_ntpi	30	jun	1986	1	jul 1986
vp9	ready_alert	1	jul	1986	31	jul 1986
vp48	natops	7	jul	1986	18	jul 1986
vp47	ntpi	14	jul	1986	15	jul 1986
vp47	natops	21	jul	1986	1	aug 1986
vp48	ready_alert	1	aug	1986	31	aug 1986
vp9	pre_mrci	5	aug	1986	8	aug 1986
vp9	mrci	18	aug	1986	21	aug 1986
vp47	ready_alert	1	sep	1986	30	sep 1986
vp50	pre_mrci	2	sep	1986	5	sep 1986
vp19	natops	8	sep	1986	19	sep 1986
vp9	command	29	sep	1986	3	oct 1986
vp50	mrci	30	sep	1986	3	oct 1986
vp50	mrci	30	sep	1986	3	oct 1986

LIST OF REFERENCES

1. Ashour, S., *Sequencing Theory*, pp. 1-11, Springer-Verlag, Berlin, Germany, 1972.

2. O'Brien, J.J., *Scheduling Handbook*, pp. 2-5, McGraw-Hill Book Company, New York, 1969.

3. Bellman, R., Esogbue, A.O., & Nabeshima, I., *Mathematical Aspects of Scheduling & Applications*, Pergamon Press, New York, 1982.

4. Rowe, N.C., *Artificial Intelligence Through Prolog*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

5. Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P., Optimization by simulated annealling, *Science*, Volume 220, pp. 671-680, May 13, 1983.

6. Goehring, D.J., "Time Constrained Planning Using Simulated Annealing", International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems Proceedings, v. 2, pp. 1066-1070, *Association for Computing Machinery*, New York, 1988.

7. Hutson, D.V., *A Program for Scheduling a Patrol Wing Training Plan*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1988.

8. Interview between Larry M. Larson, Lieutenant Commander, USN, Weapons Training Officer, Commander Patrol Wing Ten, NAS Moffett Field, California, and author, April 18, 1989.

9. Interview between Chalker W. Brown, Commander, USN, Training Officer, Commander Patrol Wing Ten, NAS Moffett Field, California, and author, November 18, 1988.

10. Interview between Robert M. Dawson, Lieutenant Commander, USN, Assistant Training Officer, Commander Patrol Wing Ten, NAS Moffett Field, California, and author, April 18, 1989.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
5.	Curricular Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
6.	Associate Professor Neil C. Rowe, Code 52Rp Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	2
7.	Assistant Professor Yuh-jeng Lee, Code 52Le Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1

8.	Commanding Officer Patrol Squadron TWENTY-TWO Naval Air Station Barbers Point, Hawaii 96862-6100	1
9.	LCDR George S. Sanford, Jr. Weapons Training Officer Patrol Wings Pacific Naval Air Station Moffett Field, California 94035	1
10.	CDR Chalker M. Brown Training Officer Patrol Wing TEN Naval Air Station Moffett Field, California 94035-5022	1
11.	Dr. Hank Smith Education Coordinator Patrol Squadron THIRTY-ONE Naval Air Station Moffett Field, California 94035	1
12.	LCDR Robert D. Powell 2317 Georgetown Drive Bartlesville, Oklahoma 74003	2